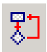
Lập trình
Lập trình có mục đích để làm dễ dàng và đơn giản hoá nhiệm vụ tính toán. Để lập trình trên Mathcad phải dùng đến ngôn ngữ Programming được xây dựng trong Mathcad bao gồm: nhánh điều kiện (IF), cấu trúc vòng lập (FOR, WHILE…), trình bày lỗi…

Câu điều kiện "if"

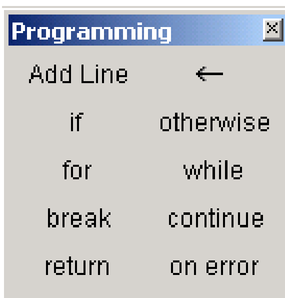
Thực hiện theo các bước sau:

Kích ngay bên phải khung nhập biểu thức nơi muốn chèn câu lệnh "if"

Từ thanh Math : Kích vào biểu tượng



, xuất hiện hộp thoại Programming (hìn.h7. 1)



Hình 7.1. Programming

Kích vào nút Add Line, xuất hiện ***SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.***

Tại vị trí khung trống ở trên kích vào nút "if" (hoặc từ bàn phím nhấn Shift+]), ngay bên phải khung nhập biểu thức, nhập biểu thức Boolean.

Tại vị trí khung trống ở dưới kích vào nút "otherwise", gõ giá trị muốn chương trình trả về nếu kết quả điều kiện là sai.

Ví dụ:

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

Lưu ý: Nếu sử dụng nhiều câu lệnh “if” trước câu lệnh “otherwise” thì câu lệnh “otherwise” chỉ được thực hiện khi tất cả các điều kiện đều sai.

CÂU LỆNH VÒNG LẶP (PROGRAM LOOPS)

Loop là lệnh chương trình được dùng để làm cho một hay nhiều câu lệnh (nội dung vòng lặp) điều hoạt theo chu trình cho đến khi thoả mãn điều kiện đã đưa ra. Trong Mathcad có hai loại câu lệnh lặp:

Câu lệnh “for”

Câu lệnh “for” : được áp dụng khi bạn biết chính xác số lần vòng lặp được thực thi.

Để thực hiện vòng lặp “for” tiến hành theo các bước sau:

Kích ngay bên phải khung nhập biểu thức nơi muốn chèn câu lệnh “for”

```
sum(n) :=  | s ← 0
           | ■
```

Kích vào nút “for” trên thanh Math (hoặc từ bàn phím nhấn Ctrl+”)

```
sum(n) :=  | s ← 0
           | for ■ ∈ ■
           |    ■
```

Bên trái

$\in$

gõ biến thay đổi, bên phải

$\in$

nhập dãy số chạy

```
sum(n) := | s ← 0
          | for  x ∈ 1 .. n
          |   ■
```

Nhập biểu thức vào khung trống bên dưới

```
sum(n) := | s ← 0
          | for  x ∈ 1 .. n
          |   s ← s + 1
```

Ví dụ 1: Tính tổng của n số nguyên dương đầu tiên

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

Ví dụ 2: Tính giai thừa của một số

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

Câu lệnh “while”

Câu lệnh “while” : được dùng khi bạn muốn vòng lặp dừng lại theo điều kiện hiện hành nhưng lại không biết chính xác khi nào điều kiện đó

xảy ra.

Khi dùng các câu lệnh lặp, bạn cần phải cắt chúng ra thành từng quy trình hoặc kiểm soát tính hoạt động của câu lệnh.

Ví dụ:

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

Câu lệnh ngắt “break”

Câu lệnh “break” : trong vòng lặp “loop” khi muốn dừng quy trình điều hoạt câu lệnh lặp.

- Kích vào khung nhập chương trình trong đó muốn đặt câu lệnh “break”

```
■  if  x ≥ 8
y
```

Kích vào nút “break” trên thanh Math (hoặc từ bàn phím nhấn Ctrl+{)

```
break   if  x ≥ 8
y
```

Khi Mathcad bắt gặp câu lệnh “break” trong phần thân của vòng lặp “for” hoặc “while”:

Chu trình lặp sẽ ngưng sự điều hoạt và trả về giá trị đã được tính sau cùng.

Sau đó, chương trình sẽ tiếp tục điều hoạt ngay dòng kế tiếp của chương trình sau chu trình.

Ví dụ:

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

Hiện kết quả của chương trình “return”

Câu lệnh “return” : theo mặc định, chương trình trả lại những gì nằm trên dòng cuối cùng. Tuy nhiên, có thể trả lại giá trị ở một nơi nào đó trong chương trình với câu lệnh “return”.

Để chèn câu lệnh “return”, thực hiện như sau:

Kích vào khung nhập chương trình trong đó muốn đặt câu lệnh “return”

```
if  x=0
    | a ← 2
    | ▪
for  i ∈ 0 .. x
  t ← t + x
t
```

Kích vào nút “return” trên thanh Math (hoặc từ bàn phím nhấn Ctrl+|)

Ví dụ:

```
if  x=0
    | a ← 2
    | return ▪
for  i ∈ 0 .. x
  t ← t + x
t
```

Trong vùng trống bên phải câu lệnh “return”, nhập những gì bạn muốn trả về. Các câu lệnh “return” rất hữu dụng khi bạn muốn trả về giá trị từ vòng lặp.

```
x := 0
| if  x=0          = 2
|   | a ← 2
|   | return  a
| for  i ∈ 0 .. x
|   t ← t + x
| t
```

Tìm lỗi chương trình

Câu lệnh “on error” : muốn trả về giá trị cần giải quyết khi Mathcad bắt gặp lỗi trong chương trình

Để chèn câu lệnh “on error”, thực hiện như sau:

Kích vào khung nhập chương trình trong đó muốn đặt câu lệnh “on error”

```
f(x) := ▪
```

Kích vào nút “on error” trên thanh Math (hoặc từ bàn phím nhấn Ctrl+’)

```
f(x) := ▪  on error  ▪
```

Trong vùng trống bên phải câu lệnh “on error”, nhập những gì bạn muốn trả về.

$$f(x) := \blacksquare \quad \text{on error} \quad \frac{1}{2 - x}$$

Trong vùng trống bên trái câu lệnh “on error”, nhập những gì bạn muốn trả về nếu biểu thức mặc định không thể tính được. Dùng nút lệnh “Add Line” để chèn những khung nhập lệnh bổ sung.

Ví dụ:

$$f(x) := \infty \quad \text{on error} \quad \frac{1}{2 - x}$$

Biểu thức bên phải sẽ được tính và được trả về nếu không có lỗi xảy ra. Và nếu có lỗi xảy ra, biểu thức bên trái sẽ được trả về.

BÀI TẬP CHƯƠNG 7

1.

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

Giới thiệu về lập trình hướng đối tượng
Phần này trình bày về lập trình hướng đối tượng

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (OOP) LÀ GÌ ?

Lập trình hướng đối tượng (Object-Oriented Programming, viết tắt là OOP) là một phương pháp mới trên bước đường tiến hóa của việc lập trình máy tính, nhằm làm cho chương trình trở nên linh hoạt, tin cậy và dễ phát triển. Tuy nhiên để hiểu được OOP là gì, chúng ta hãy bắt đầu từ lịch sử của quá trình lập trình – xem xét OOP đã tiến hóa như thế nào.

Lập trình tuyến tính

Máy tính đầu tiên được lập trình bằng mã nhị phân, sử dụng các công tắt cơ khí để nạp chương trình. Cùng với sự xuất hiện của các thiết bị lưu trữ lớn và bộ nhớ máy tính có dung lượng lớn nên các ngôn ngữ lập trình cấp cao đầu tiên được đưa vào sử dụng . Thay vì phải suy nghĩ trên một dãy các bit và byte, lập trình viên có thể viết một loạt lệnh gần với tiếng Anh và sau đó chương trình dịch thành ngôn ngữ máy.

Các ngôn ngữ lập trình cấp cao đầu tiên được thiết kế để lập các chương trình làm các công việc tương đối đơn giản như tính toán. Các chương trình ban đầu chủ yếu liên quan đến tính toán và không đòi hỏi gì nhiều ở ngôn ngữ lập trình. Hơn nữa phần lớn các chương trình này tương đối ngắn, thường ít hơn 100 dòng.

Khi khả năng của máy tính tăng lên thì khả năng để triển khai các chương trình phức tạp hơn cũng tăng lên. Các ngôn ngữ lập trình ngày trước không còn thích hợp đối với việc lập trình đòi hỏi cao hơn. Các phương tiện cần thiết để sử dụng lại các phần mã chương trình đã viết hầu như không có trong ngôn ngữ lập trình tuyến tính. Thật ra, một đoạn lệnh thường phải được chép lặp lại mỗi khi chúng ta dùng trong nhiều chương trình do đó chương trình dài dòng, logic của chương trình khó hiểu. Chương trình được điều khiển để nhảy đến nhiều chỗ mà thường không có sự giải thích rõ ràng, làm thế nào để chương trình đến chỗ cần thiết hoặc tại sao như vậy.

Ngôn ngữ lập trình tuyến tính không có khả năng kiểm soát phạm vi nhìn thấy của các dữ liệu. Mọi dữ liệu trong chương trình đều là dữ liệu toàn cục nghĩa là chúng có thể bị sửa đổi ở bất kỳ phần nào của chương trình. Việc dò tìm các thay đổi không mong muốn đó của các phần tử dữ liệu trong một dãy mã lệnh dài và vòng vèo đã từng làm cho các lập trình viên rất mất thời gian.

Lập trình cấu trúc

Rõ ràng là các ngôn ngữ mới với các tính năng mới cần phải được phát triển để có thể tạo ra các ứng dụng tinh vi hơn. Vào cuối các năm trong 1960 và 1970, ngôn ngữ lập trình có cấu trúc ra đời. Các chương trình có cấu trúc được tổ chức theo các công việc mà chúng thực hiện.

Về bản chất, chương trình chia nhỏ thành các chương trình con riêng rẽ (còn gọi là hàm hay thủ tục) thực hiện các công việc rời rạc trong quá trình lớn hơn, phức tạp hơn. Các hàm này được giữ càng độc lập với nhau càng nhiều càng tốt, mỗi hàm có dữ liệu và logic riêng.Thông tin được chuyển giao giữa các hàm thông qua các tham số, các hàm có thể có các biến cục bộ mà không một ai nằm bên ngoài phạm vi của hàm lại có thể truy xuất được chúng. Như vậy, các hàm có thể được xem là các chương trình con được đặt chung với nhau để xây dựng nên một ứng dụng.

Mục tiêu là làm sao cho việc triển khai các phần mềm dễ dàng hơn đối với các lập trình viên mà vẫn cải thiện được tính tin cậy và dễ bảo quản chương trình. Một chương trình có cấu trúc được hình thành bằng cách bẻ gãy các chức năng cơ bản của chương trình thành các mảnh nhỏ mà sau đó trở thành các hàm. Bằng cách cô lập các công việc vào trong các hàm, chương trình có cấu trúc có thể làm giảm khả năng của một hàm này ảnh hưởng đến một hàm khác. Việc này cũng làm cho việc tách các vấn đề trở nên dễ dàng hơn. Sự gói gọn này cho phép chúng ta có thể viết các chương trình sáng sủa hơn và giữ được điều khiển trên từng hàm. Các biến toàn cục không còn nữa và được thay thế bằng các tham số và biến cục bộ có phạm vi nhỏ hơn và dễ kiểm soát hơn. Cách tổ chức tốt hơn này nói lên rằng chúng ta có khả năng quản lý logic của cấu

trúc chương trình, làm cho việc triển khai và bảo dưỡng chương trình nhanh hơn và hữu hiện hơn và hiệu quả hơn.

Một khái niệm lớn đã được đưa ra trong lập trình có cấu trúc là sự trừu tượng hóa (Abstraction). Sự trừu tượng hóa có thể xem như khả năng quan sát một sự việc mà không cần xem xét đến các chi tiết bên trong của nó. Trong một chương trình có cấu trúc, chúng ta chỉ cần biết một hàm đã cho có thể làm được một công việc cụ thể gì là đủ. Còn làm thế nào mà công việc đó lại thực hiện được là không quan trọng, chừng nào hàm còn tin cậy được thì còn có thể dùng nó mà không cần phải biết nó thực hiện đúng đắn chức năng của mình như thế nào. Điều này gọi là sự trừu tượng hóa theo chức năng (Functional abstraction) và là nền tảng của lập trình có cấu trúc.

Ngày nay, các kỹ thuật thiết kế và lập trình có cấu trúc được sử rộng rãi. Gần như mọi ngôn ngữ lập trình đều có các phương tiện cần thiết để cho phép lập trình có cấu trúc. Chương trình có cấu trúc dễ viết, dễ bảo dưỡng hơn các chương trình không cấu trúc.

Sự nâng cấp như vậy cho các kiểu dữ liệu trong các ứng dụng mà các lập trình viên đang viết cũng đang tiếp tục diễn ra. Khi độ phức tạp của một chương trình tăng lên, sự phụ thuộc của nó vào các kiểu dữ liệu cơ bản mà nó xử lý cũng tăng theo. Vấn đề trở rõ ràng là cấu trúc dữ liệu trong chương trình quan trọng chẳng kém gì các phép toán thực hiện trên chúng. Điều này càng trở rõ ràng hơn khi kích thước của chương trình càng tăng. Các kiểu dữ liệu được xử lý trong nhiều hàm khác nhau bên trong một chương trình có cấu trúc. Khi có sự thay đổi trong các dữ liệu này thì cũng cần phải thực hiện cả các thay đổi ở mọi nơi có các thao tác tác động trên chúng. Đây có thể là một công việc tốn thời gian và kém hiệu quả đối với các chương trình có hàng ngàn dòng lệnh và hàng trăm hàm trở lên.

Một yếu điểm nữa của việc lập trình có cấu trúc là khi có nhiều lập trình viên làm việc theo nhóm cùng một ứng dụng nào đó. Trong một chương trình có cấu trúc, các lập trình viên được phân công viết một tập hợp các hàm và các kiểu dữ liệu. Vì có nhiều lập trình viên khác nhau quản lý các

hàm riêng, có liên quan đến các kiểu dữ liệu dùng chung nên các thay đổi mà lập trình viên tạo ra trên một phần tử dữ liệu sẽ làm ảnh hưởng đến công việc của tất cả các người còn lại trong nhóm. Mặc dù trong bối cảnh làm việc theo nhóm, việc viết các chương trình có cấu trúc thì dễ dàng hơn nhưng sai sót trong việc trao đổi thông tin giữa các thành viên trong nhóm có thể dẫn tới hậu quả là mất rất nhiều thời gian để sửa chữa chương trình.

Sự trừu tượng hóa dữ liệu

Sự trừu tượng hóa dữ liệu (Data abstraction) tác động trên các dữ liệu cũng tương tự như sự trừu tượng hóa theo chức năng. Khi có trừu tượng hóa dữ liệu, các cấu trúc dữ liệu và các phần tử có thể được sử dụng mà không cần bận tâm đến các chi tiết cụ thể. Chẳng hạn như các số dấu chấm động đã được trừu tượng hóa trong tất cả các ngôn ngữ lập trình, Chúng ta không cần quan tâm cách biểu diễn nhị phân chính xác nào cho số dấu chấm động khi gán một giá trị, cũng không cần biết tính bất thường của phép nhân nhị phân khi nhân các giá trị dấu chấm động. Điều quan trọng là các số dấu chấm động hoạt động đúng đắn và hiểu được.

Sự trừu tượng hóa dữ liệu giúp chúng ta không phải bận tâm về các chi tiết không cần thiết. Nếu lập trình viên phải hiểu biết về tất cả các khía cạnh của vấn đề, ở mọi lúc và về tất cả các hàm của chương trình thì chỉ ít hàm mới được viết ra, may mắn thay trừu tượng hóa theo dữ liệu đã tồn tại sẵn trong mọi ngôn ngữ lập trình đối với các dữ liệu phức tạp như số dấu chấm động. Tuy nhiên chỉ mới gần đây, người ta mới phát triển các ngôn ngữ cho phép chúng ta định nghĩa các kiểu dữ liệu trừu tượng riêng.

Lập trình hướng đối tượng

Khái niệm hướng đối tượng được xây dựng trên nền tảng của khái niệm lập trình có cấu trúc và sự trừu tượng hóa dữ liệu. Sự thay đổi căn bản ở chỗ, một chương trình hướng đối tượng được thiết kế xoay quanh dữ liệu mà chúng ta có thể làm việc trên đó, hơn là theo bản thân chức năng

của chương trình. Điều này hoàn toàn tự nhiên một khi chúng ta hiểu rằng mục tiêu của chương trình là xử lý dữ liệu. Suy cho cùng, công việc mà máy tính thực hiện vẫn thường được gọi là xử lý dữ liệu. Dữ liệu và thao tác liên kết với nhau ở một mức cơ bản (còn có thể gọi là mức thấp), mỗi thứ đều đòi hỏi ở thứ kia có mục tiêu cụ thể, các chương trình hướng đối tượng làm tường minh mối quan hệ này.

Lập trình hướng đối tượng (Object Oriented Programming - gọi tắt là OOP) hay chi tiết hơn là Lập trình định hướng đối tượng, chính là phương pháp lập trình lấy đối tượng làm nền tảng để xây dựng thuật giải, xây dựng chương trình. Thực chất đây không phải là một phương pháp mới mà là một cách nhìn mới trong việc lập trình. Để phân biệt, với phương pháp lập trình theo kiểu cấu trúc mà chúng ta quen thuộc trước đây, hay còn gọi là phương pháp lập trình hướng thủ tục (Procedure-Oriented Programming), người lập trình phân tích một nhiệm vụ lớn thành nhiều công việc nhỏ hơn, sau đó dần dần chi tiết, cụ thể hoá để được các vấn đề đơn giản, để tìm ra cách giải quyết vấn đề dưới dạng những thuật giải cụ thể rõ ràng qua đó dễ dàng minh hoạ bằng ngôn ngữ giải thuật (hay còn gọi các thuật giải này là các chương trình con). Cách thức phân tích và thiết kế như vậy chúng ta gọi là nguyên lý lập trình từ trên xuống (top-down), để thể hiện quá trình suy diễn từ cái chung cho đến cái cụ thể.

Các chương trình con là những chức năng độc lập, sự ghép nối chúng lại với nhau cho chúng ta một hệ thống chương trình để giải quyết vấn đề đặt ra. Chính vì vậy, cách thức phân tích một hệ thống lấy chương trình con làm nền tảng, chương trình con đóng vai trò trung tâm của việc lập trình, được hiểu như phương pháp lập trình hướg về thủ tục. Tuy nhiên, khi phân tích để thiết kế một hệ thống không nhất thiết phải luôn luôn suy nghĩ theo hướng “làm thế nào để giải quyết công việc”, chúng ta có thể định hướng tư duy theo phong cách “với một số đối tượng đã có, phải làm gì để giải quyết được công việc đặt ra” hoặc phong phú hơn, “làm cái gì với một số đối tượng đã có đó”, từ đó cũng có thể giải quyết được những công việc cụ thể. Với phương pháp phân tích trong đó đối tượng đóng vai trò trùng tâm của việc lập trình như vậy, người ta gọi là nguyên lý lập trình từ dưới lên (Bôttm-up).

Lập trình hướng đối tượng liên kết cấu trúc dữ liệu với các thao tác, theo cách mà tất cả thường nghĩ về thế giới quanh mình. Chúng ta thường gắn một số các hoạt động cụ thể với một loại hoạt động nào đó và đặt các giả thiết của mình trên các quan hệ đó.

Ví dụ1.1: Để dễ hình dùng hơn, chúng ta thủ nhìn qua các công trình xây dựng hiện đại, như sân vận động có mái che hình vòng cung, những kiến trúc thẩm mĩ với đường nét hình cong. Tất cả những sản phẩm đó xuất hiện cùng với những vật liệu xây dựng. Ngày nay, không chỉ chồng lên nhau những viên gạch, những tảng đá để tạo nên những quần thể kiến trúc (như Tháp Chàm Nha Trang, Kim Tự Tháp,...), mà có thể với bêtông, sắt thép và không nhiều lắm những viên gạch, người xây dựng cũng có thể thiết kế những công trình kiến trúc tuyệt mỹ, những toà nhà hiện đại. Chính các chất liệu xây dựng đã làm ảnh hưởng phương pháp xây dựng, chất liệu xây dựng và nguyên lý kết dính caá chất liệu đó lại với nhau cho chúng ta một đối tượng để khảo sát, Chất liệu xây dựng và nguyên lý kết dính các chất liệu lại với nhau được hiểu theo nghĩa dữ liệu và chương trình con tác động trên dữ liệu đó.

Ví dụ1.2: Chúng ta biết rằng một chiếc xe có các bánh xe, di chuyển được và có thể đổi hướng của nó bằng cách quẹo tay lái. Tương tự như thế, một cái cây là một loại thực vật có thân gỗ và lá. Một chiếc xe không phải là một cái cây, mà cái cây không phải là một chiếc xe, chúng ta có thể giả thiết rằng cái mà chúng ta có thể làm được với một chiếc xe thì không thể làm được với một cái cây. Chẳng hạn, thật là vô nghĩa khi muốn lái một cái cây, còn chiếc xe thì lại chẳng lớn thêm được khi chúng ta tưới nước cho nó.

Lập trình hướng đối tượng cho phép chúng ta sử dụng các quá trình suy nghĩ như vậy với các khái niệm trừu tượng được sử dụng trong các chương trình máy tính. Một mẫu tin (record) nhân sự có thể được đọc ra, thay đổi và lưu trữ lại; còn số phức thì có thể được dùng trong các tính toán. Tuy vậy không thể nào lại viết một số phức vào tập tin làm mẫu tin nhân sự và ngược lại hai mẫu tin nhân sự lại không thể cộng với nhau được. Một chương trình hướng đối tượng sẽ xác định đặc điểm và hành vi cụ thể của các kiểu dữ liệu, điều đó cho phép chúng ta biết một cách

chính xác rằng chúng ta có thể có được những gì ở các kiểu dữ liệu khác nhau.

Chúng ta còn có thể tạo ra các quan hệ giữa các kiểu dữ liệu tương tự nhưng khác nhau trong một chương trình hướng đối tượng. Người ta thường tự nhiên phân loại ra mọi thứ, thường đặt mối liên hệ giữa các khái niệm mới với các khái niệm đã có, và thường có thể thực hiện suy diễn giữa chúng trên các quan hệ đó. Hãy quan niệm thế giới theo kiểu cấu trúc cây, với các mức xây dựng chi tiết hơn kế tiếp nhau cho các thế hệ sau so với các thế hệ trước. Đây là phương pháp hiệu quả để tổ chức thế giới quanh chúng ta. Các chương trình hướng đối tượng cũng làm việc theo một phương thức tương tự, trong đó chúng cho phép xây dựng các các cơ cấu dữ liệu và thao tác mới dựa trên các cơ cấu có sẵn, mang theo các tính năng của các cơ cấu nền mà chúng dựa trên đó, trong khi vẫn thêm vào các tính năng mới.

Lập trình hướng đối tượng cho phép chúng ta tổ chức dữ liệu trong chương trình theo một cách tương tự như các nhà sinh học tổ chức các loại thực vật khác nhau. Theo cách nói lập trình đối tượng, xe hơi, cây cối, các số phức, các quyển sách đều được gọi là các lớp (Class).

Một lớp là một bản mẫu mô tả các thông tin cấu trúc dữ liệu, lẫn các thao tác hợp lệ của các phần tử dữ liệu. Khi một phần tử dữ liệu được khai báo là phần tử của một lớp thì nó được gọi là một đối tượng (Object). Các hàm được định nghĩa hợp lệ trong một lớp được gọi là các phương thức (Method) và chúng là các hàm duy nhất có thể xử lý dữ liệu của các đối tượng của lớp đó. Một thực thể (Instance) là một vật thể có thực bên trong bộ nhớ, thực chất đó là một đối tượng (nghĩa là một đối tượng được cấp phát vùng nhớ).

Mỗi một đối tượng có riêng cho mình một bản sao các phần tử dữ liệu của lớp còn gọi là các biến thực thể (Instance variable). Các phương thức định nghĩa trong một lớp có thể được gọi bởi các đối tượng của lớp đó. Điều này được gọi là gửi một thông điệp (Message) cho đối tượng. Các thông điệp này phụ thuộc vào đối tượng, chỉ đối tượng nào nhận thông điệp mới phải làm việc theo thông điệp đó. Các đối tượng đều độc lập

với nhau vì vậy các thay đổi trên các biến thể hiện của đối tượng này không ảnh hưởng gì trên các biến thể hiện của các đối tượng khác và việc gửi thông điệp cho một đối tượng này không ảnh hưởng gì đến các đối tượng khác.

Như vậy, đối tợng được hiểu theo nghĩa là một thực thể mà trong đó caá dữ liệu và thủ tục tác động lên dữ liệu đã được đóng gói lại với nhau. Hay "đối tượng được đặc trưng bởi một số thao tác (operation) và các thông tin (information) ghi nhơ sự tác động của caá thao tác này."

Ví dụ 1.3: Khi nghiên cứ về ngăn xếp (stack), ngoài các dữ liệu vùng chứa ngăn xếp, đỉnh của ngăn xếp, chúng ta phải cài đặt kèm theo các thao tác như khởi tạo (creat) ngăn xếp, kiểm tra ngăn xếp rỗng (empty), đẩy (push) một phần tử vào ngăn xếp, lấy (pop) một phần tử ra khỏi ngăn xếp. Trên quan điểm lấy đối tượng làm nền tảng, rõ ràng dữ liệu và các thao tác trên dữ liệu luôn gắn bó với nhau, sự kết dính chúng chính là đối tượng chúng ta cần khảo sát.

Các thao tác trong đối tượng được gọi là các phương thức hay hành vi của đối tượng đó. Phương thức và dữ liệu của đối tượng luôn tác động lẫn nhau và có vai trò ngang nhau trong đối tượng, Phương thức của đối tượng được qui định bởi dữ liệu và ngược lại, dữ liệu của đối tượng được đặt trưng bởi các phương thức của đối tượng. Chính nhờ sự gắn bó đó, chúng ta có thể gởi cùng một thông điệp đến những đối tượng khác nhau. Điều này giúp người lập trình không phải xử lý trong chương trình của mình một dãy các cấu trúc điều khiển tuỳ theo thông điệp nhận vào, mà chương trình được xử lý vào thời điểm thực hiện.

Tóm lại, so sánh lập trình cấu trúc với chương trình con làm nền tảng:

Chương trình = Cấu trúc dữ liệu + Thuật giải

Trong lập trình hướng đối tượng chúng ta có:

Đối tượng = Phương thức + Dữ liệu

Đây chính là 2 quan điểm lập trình đang tồn tại và phát triển trong thế giới ngày nay.

MỘT SỐ KHÁI NIỆM MỚI TRONG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Trong phần này, chúng ta tìm hiểu các khái niệm như sự đóng gói, tính kế thừa và tính đa hình. Đây là các khái niệm căn bản, là nền tảng tư tưởng của lập trình hướng đối tượng. Hiểu được khái niệm này, chúng ta bước đầu tiếp cận với phong cách lập trình mới, phong cách lập trình dựa vào đối tượng làm nền tảng mà trong đó quan điểm che dấu thông tin thông qua sư đóng gói là quan điểm trung tâm của vấn đề.

Sự đóng gói (Encapsulation)

Sự đóng gói là cơ chế ràng buộc dữ liệu và thao tác trên dữ liệu đó thành một thể thống nhất, tránh được các tác động bất ngờ từ bên ngoài. Thể thống nhất này gọi là đối tượng.

Trong Objetc Oriented Software Engineering của Ivar Jacibson, tất cả các thông tin của một hệ thống định hướng đối tượng được lưu trữ bên trong đối tượng của nó và chỉ có thể hành động khi các đối tượng đó được ra lệnh thực hiện các thao tác. Như vật, sự đóng gói không chỉ đơn thuần là sự gom chung dữ liệu và chương trình vào trong một khối, chúng còn được hiểu theo nghĩa là sự đồng nhất giữa dữ liệu và các thao tác tác động lên dữ liệu đó.

Trong một đối tượng, dữ liệu hay thao tác hay cả hai có thể là riêng (private) hoặc chung (public) của đối tượng đó. Thao tác hay dữ liệu riêng là thuộc về đối tượng đó chỉ được truy cập bởi các thành phần của đối tượng, điều này nghĩa là thao tác hay dữ liệu riêng không thể truy cập bởi các phần khác của chương trình tồn tại ngoài đối tượng. Khi thao tác hay dữ liệu là chung, các phần khác của chương trình có thể truy cập nó mặc dù nó được định nghĩa trong một đối tượng. Các thành phần chung

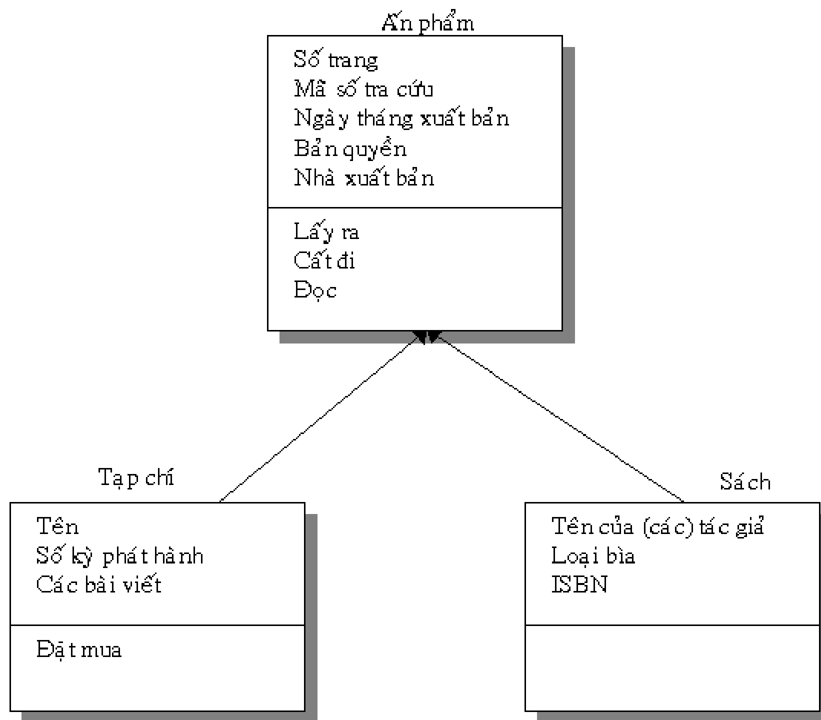
của một đối tượng dùng để cung cấp một giao diện có điều khiển cho các thành thành riêng của đối tượng.

Cơ chế đóng gói là phương thức tốt để thực hiện cơ chế che dấu thông tin so với các ngôn ngữ lập trình cấu trúc.

Tính kế thừa (Inheritance)

Chúng ta có thể xây dựng các lớp mới từ các lớp cũ thông qua sự kế thừa. Một lớp mới còn gọi là lớp dẫn xuất (derived class), có thể thừa hưởng dữ liệu và các phương thức của lớp cơ sở (base class) ban đầu. Trong lớp này, có thể bổ sung các thành phần dữ liệu và các phương thức mới vào những thành phần dữ liệu và các phương thức mà nó thừa hưởng từ lớp cơ sở. Mỗi lớp (kể cả lớp dẫn xuất) có thể có một số lượng bất kỳ các lớp dẫn xuất. Qua cơ cấu kế thừa này, dạng hình cây của các lớp được hình thành. Dạng cây của các lớp trông giống như các cây gia phả vì thế các lớp cơ sở còn được gọi là lớp cha (parent class) và các lớp dẫn xuất được gọi là lớp con (child class).

Ví dụ 1.2: Chúng ta sẽ xây dựng một tập các lớp mô tả cho thư viện các ấn phẩm. Có hai kiểu ấn phẩm: tạp chí và sách. Chúng ta có thể tạo một ấn phẩm tổng quát bằng cách định nghĩa các thành phần dữ liệu tương ứng với số trang, mã số tra cứu, ngày tháng xuất bản, bản quyền và nhà xuất bản. Các ấn phẩm có thể được lấy ra, cất đi và đọc. Đó là các phương thức thực hiện trên một ấn phẩm. Tiếp đó chúng ta định nghĩa hai lớp dẫn xuất tên là tạp chí và sách. Tạp chí có tên, số ký phát hành và chứa nhiều bài của các tác giả khác nhau . Các thành phần dữ liệu tương ứng với các yếu tố này được đặt vào định nghĩa của lớp tạp chí. Tạp chí cũng cần có một phương thức nữa đó là đặt mua. Các thành phần dữ liệu xác định cho sách sẽ bao gồm tên của (các) tác giả, loại bìa (cứng hay mềm) và số hiệu ISBN của nó. Như vậy chúng ta có thể thấy, sách và tạp chí có chung các đặc trưng ấn phẩm, trong khi vẫn có các thuộc tính riêng của chúng.



Hình 1.1: Lớp ấn phẩm và các lớp dẫn xuất của nó.

Với tính kế thừa, chúng ta không phải mất công xây dựng lại từ đầu các lớp mới, chỉ cần bổ sung để có được trong các lớp dẫn xuất các đặc trưng cần thiết.

Tính đa hình (Polymorphism)

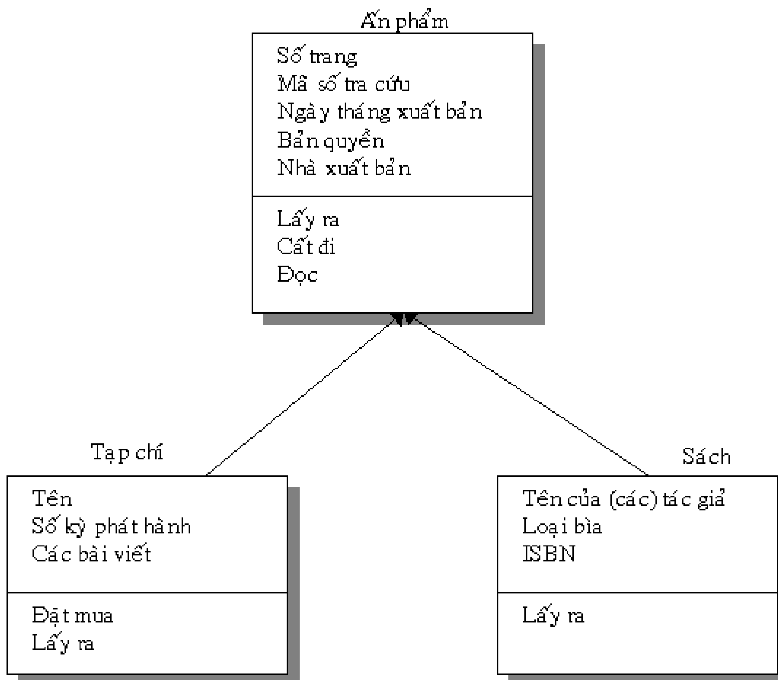
Đó là khả năng để cho một thông điệp có thể thay đổi cách thực hiện của nó theo lớp cụ thể của đối tượng nhận thông điệp. Khi một lớp dẫn xuất được tạo ra, nó có thể thay đổi cách thực hiện các phương thức nào đó mà nó thừa hưởng từ lớp cơ sở của nó. Một thông điệp khi được gởi đến một đối tượng của lớp cơ sở, sẽ dùng phương thức đã định nghĩa cho nó trong lớp cơ sở. Nếu một lớp dẫn xuất định nghĩa lại một phương thức thừa hưởng từ lớp cơ sở của nó thì một thông điệp có cùng tên với phương thức này, khi được gởi tới một đối tượng của lớp dẫn xuất sẽ gọi phương thức đã định nghĩa cho lớp dẫn xuất.

Như vậy đa hình là khả năng cho phép gởi cùng một thông điệp đến những đối tượng khác nhau có cùng chung một đặc điểm, nói cách khác

thông điệp được gởi đi không cần biết thực thể nhận thuộc lớp nào, chỉ biết rằng tập hợp các thực thể nhận có chung một tính chất nào đó. Chẳng hạn, thông điệp “vẽ hình” được gởi đến cả hai đối tượng hình hộp và hình tròn. Trong hai đối tượng này đều có chung phương thức vẽ hình, tuy nhiên tuỳ theo thời điểm mà đối tượng nhận thông điệp, hình tương ứng sẽ được vẽ lên.

Trong các ngôn ngữ lập trình OOP, tính đa hình thể hiện qua khả năng cho phép mô tả những phương thức có tên giống nhau trong các lớp khác nhau. Đặc điểm này giúp người lập trình không phải viết những cấu trúc điều khiển rườm rà trong chương trình, các khả năng khác nhau của thông điệp chỉ thực sự đòi hỏi khi chương trình thực hiện.

Ví dụ 1.3: Xét lại ví dụ 1.2, chúng ta thấy rằng cả tạp chí và và sách đều phải có khả năng lấy ra. Tuy nhiên phương pháp lấy ra cho tạp chí có khác so với phương pháp lấy ra cho sách, mặc dù kết quả cuối cùng giống nhau. Khi phải lấy ra tạp chí, thì phải sử dụng phương pháp lấy ra riêng cho tạp chí (dựa trên một bản tra cứu) nhưng khi lấy ra sách thì lại phải sử dụng phương pháp lấy ra riêng cho sách (dựa trên hệ thống phiếu lưu trữ). Tính đa hình cho phép chúng ta xác định một phương thức để lấy ra một tạp chí hay một cuốn sách. Khi lấy ra một tạp chí nó sẽ dùng phương thức lấy ra dành riêng cho tạp chí, còn khi lấy ra một cuốn sách thì nó sử dụng phương thức lấy ra tương ứng với sách. Kết quả là chỉ cần một tên phương thức duy nhất được dùng cho cả hai công việc tiến hành trên hai lớp dẫn xuất có liên quan, mặc dù việc thực hiện của phương thức đó thay đổi tùy theo từng lớp.



Tính đa hình dựa trên sự nối kết (Binding), đó là quá trình gắn một phương thức với một hàm thực sự. Khi các phương thức kiểu đa hình được sử dụng thì trình biên dịch chưa thể xác định hàm nào tương ứng với phương thức nào sẽ được gọi. Hàm cụ thể được gọi sẽ tuỳ thuộc vào việc phần tử nhận thông điệp lúc đó là thuộc lớp nào, do đó hàm được gọi chỉ xác định được vào lúc chương trình chạy. Điều này gọi là sự kết nối muộn (Late binding) hay kết nối lúc chạy (Runtime binding) vì nó xảy ra khi chương trình đang thực hiện.

Hình 1.2: Minh họa tính đa hình đối với lớp ấn phẩm và các lớp dẫn xuất của nó.

CÁC NGÔN NGỮ VÀ VÀI ỨNG DỤNG CỦA OOP

Xuất phát từ tư tưởng của ngôn ngữ SIMULA67, trung tâm nghiên cứu Palo Alto (PARC) của hãng XEROR đã tập trung 10 năm nghiên cứu để hoàn thiện ngôn ngữ OOP đầu tiên với tên gọi là Smalltalk. Sau đó các ngôn ngữ OOP lần lượt ra đời như Eiffel, Clos, Loops, Flavors, Object Pascal, Object C, C++, Delphi, Java…

Chính XEROR trên cơ sở ngôn ngữ OOP đã đề ra tư tưởng giao diện biểu tượng trên màn hình (icon base screen interface), kể từ đó Apple Macintosh cũng như Microsoft Windows phát triển giao diện đồ họa như ngày nay. Trong Microsoft Windows, tư tưởng OOP được thể hiện một cách rõ nét nhất đó là "chúng ta click vào đối tượng", mỗi đối tượng có thể là control menu, control menu box, menu bar, scroll bar, button, minimize box, maximize box, … sẽ đáp ứng công việc tùy theo đặc tính của đối tượng. Turbo Vision của hãng Borland là một ứng dụng OOP tuyệt vời, giúp lập trình viên không quan tâm đến chi tiết của chương trình gia diện mà chỉ cần thực hiện các nội dung chính của vấn đề.

Lập trình cấu trúc trong Visual Basic
Mục tiêu: Chương này giới thiệu về các cấu trúc lập trình trong VB; đây là các cấu trúc cốt lõi để xây dựng nên một chương trình VB.

Học xong chương này, sinh viên phải nắm bắt được các vấn đề sau:

- Sử dụng môi trường lập trình VB để viết mã lệnh.

- Các kiểu dữ liệu trong VB.

- Cách khai báo hằng, biến trong VB.

- Biểu thức trong VB.

- Các câu lệnh đơn cũng như các câu lệnh có cấu trúc.

- Chương trình con trong VB.

- Bẫy lỗi trong VB.

Kiến thức có liên quan:

- Cách sử dụng môi trường phát triển của VB.

Tài liệu tham khảo:

- Microsoft Visual Basic 6.0 và Lập trình Cơ sở dữ liệu - Chương 4, trang 49 - Nguyễn Thị Ngọc Mai (chủ biên), Nhà xuất bản Giáo dục - 2000.

Môi trường lập trình

Soạn thảo chương trình:

Trong Visual Basic IDE, cửa sổ mã lệnh (Code) cho phép soạn thảo chương trình. Cửa sổ này có một số chức năng nổi bật:

- Đánh dấu (Bookmarks): Chức năng này cho phép đánh dấu các dòng lệnh của chương trình trong cửa sổ mã lệnh để dễ dàng xem lại về

sau này. Để bật tắt khả năng này, chọn Bookmarks từ menu Edit, hoặc chọn từ thanh công cụ Edit.

- Các phím tắt trong cửa sổ mã lệnh:

Chöùc naêng	Phím taét
Xem cöûa soå Code	F7
Xem cöûa soå Object Browser	F2
Tìm kieám	CTRL+F
Thay theá	CTRL+H
Tìm tieáp	SHIFT+F4
Tìm ngöôïc	SHIFT+F3
Chuyeån ñeán thuû tuïc keá tieáp	CTRL+DOWN ARROW
Chuyeån ñeán thuû tuïc tröôùc ñoù	CTRL+UP ARROW
Xem ñònh nghóa	SHIFT+F2
Cuoän xuoáng moät maøn hình	CTRL+PAGE DOWN
Cuoän leân moät maøn hình	CTRL+PAGE UP
Nhaûy veà vò trí tröôùc ñoù	CTRL+SHIFT+F2
Trôû veà ñaàu cuûa moâ-ñun	CTRL+HOME

Ñeán cuoái moâ-ñun	CTRL+END

Các chức năng tự động:

- Tự động kiểm tra cú pháp (Auto Syntax Check)

Nếu chức năng này không được bật thì khi ta viết một dòng mã có chứa lỗi, VB chỉ hiển thị dòng chương trình sai với màu đỏ nhưng không kèm theo chú thích gì và tất nhiên ta có thể viết tiếp các dòng lệnh khác. Còn khi chức năng này được bật, VB sẽ cho ta biết một số thông tin về lỗi và hiển thị con trỏ ngay dòng chương trình lỗi để chờ ta sửa.

- Yêu cầu khai báo biến (Require Variable Declaration)

VB sẽ thông báo lỗi khi một biến được dùng mà không khai báo và sẽ chỉ ra vị trí của biến đó.

[missing_resource: .png]

Hình III.1: Cửa sổ Options

- Gợi nhớ mã lệnh (Code):

Khả năng Auto List Members: Tự động hiển thị danh sách các thuộc tính và phương thức của 1 điều khiển hay một đối tượng khi ta gõ vào tên của chúng. Chọn thuộc tính hay phương thức cần thao tác và nhấn phím Tab hoặc Space để đưa nó vào chương trình.

[missing_resource: .png]

Hình III.2 Cửa sổ Code với khả năng gợi nhớ Code

Kiểu dữ liệu

Khái niệm

Kiểu dữ liệu là một tập hợp các giá trị mà một biến của kiểu có thể nhận và một tập hợp các phép toán có thể áp dụng trên các giá trị đó.

Các kiểu dữ liệu cơ sở trong Visual Basic

Kiểu dữ liệu	Mô tả
Boolean	Gồm 2 giá trị: TRUE & FALSE.
Byte	Các giá trị số nguyên từ 0 – 255
Integer	Các giá trị số nguyên từ -32768 – 32767
Long	Các giá trị số nguyên từ -2147483648 – 2147483647. Kiểu dữ liệu này thường được gọi là số nguyên dài.
Single	Các giá trị số thực từ -3.402823E+38 – 3.402823E+38. Kiểu dữ liệu này còn được gọi là độ chính xác đơn.
Double	Các giá trị số thực từ -1.79769313486232E+308 - 1.79769313486232E+308. Kiểu dữ liệu này được gọi là độ chính xác kép.
Currency	Dữ liệu tiền tệ chứa các giá trị số từ -922.337.203.685.477,5808 - 922.337.203.685.477,5807.

String	Chuỗi dữ liệu từ 0 đến 65.500 ký tự hay ký số, thậm chí là các giá trị đặc biệt như ^%@. Giá trị kiểu chuỗi được đặt giữa 2 dấu ngoặc kép (“”).
Date	Dữ liệu kiểu ngày tháng, giá trị được đặt giữa cặp dấu ##. Việc định dạng hiển thị tùy thuộc vào việc thiết lập trong Control Panel.
Variant	Chứa mọi giá trị của các kiểu dữ liệu khác, kể cả mảng.

Hằng số

Khái niệm

Hằng số (Constant) là giá trị dữ liệu không thay đổi.

Khai báo hằng

[Public|Private] Const <tên hằng> [As <kiểu dữ liệu>] = <biểu thức>

Trong đó, tên hằng được đặt giống theo quy tắc đặt tên của điều khiển.

Ví dụ:

Const g = 9.8

Const Num As Integer = 4*5

Ta có thể dùng cửa sổ Object Browser để xem danh sách các hằng có sẵn của VB và VBA (Visual Basic for Application).

Trường hợp trùng tên hằng trong những thư viện khác nhau, ta có thể chỉ rõ tham chiếu hằng.

[<Libname>.][<tên mô-đun>.] <tên hằng>

Biến

Khái niệm

Biến (Variable) là vùng lưu trữ được đặt tên để chứa dữ liệu tạm thời trong quá trình tính toán, so sánh và các công việc khác.

Biến có 2 đăc điểm:

- Mỗi biến có một tên.
- Mỗi biến có thể chứa duy nhất một loại dữ liệu.

Khai báo

[Public|Private|Static|Dim] <tên biến> [As <kiểu dữ liệu>]

Trong đó, tên biến: là một tên được đặt giống quy tắc đặt tên điều khiển. Nếu cần khai báo nhiều biến trên một dòng thì mỗi khai báo cách nhau dấu phẩy (,).

Nếu khai báo biến không xác định kiểu dữ liệu thì biến đó có kiểu Variant.

Khai báo ngầm: Đây là hình thức không cần phải khai báo một biến trước khi sử dụng. Cách dùng này có vẻ thuận tiện nhưng sẽ gây một số sai sót, chẳng hạn khi ta đánh nhầm tên biến, VB sẽ hiểu đó là một biến mới dẫn đến kết quả chương trình sai mà rất khó phát hiện.

Ví dụ:

Dim Num As Long, a As Single

Dim Age As Integer

Khai báo tường minh: Để tránh rắc rối như đã nêu ở trên, ta nên quy định rằng VB sẽ báo lỗi khi gặp biến chưa được khai báo bằng dòng lệnh:

Option Explicit trong phần Declaration (khai báo) của mô-đun.

Option Explicit chỉ có tác dụng trên từng mô-đun do đó ta phải đặt dòng lệnh này trong từng mô-đun của biểu mẫu, mô-đun lớp hay mô-đun chuẩn.

Biểu thức

Khái niệm

Toán tử hay phép toán (Operator): là từ hay ký hiệu nhằm thực hiện phép tính và xử lý dữ liệu.

Toán hạng: là giá trị dữ liệu (biến, hằng…).

Biểu thức: là tập hợp các toán hạng và các toán tử kết hợp lại với nhau theo quy tắc nhất định để tính toán ra một giá trị nào đó.

Các loại phép toán

1. Các phép toán số học: Thao tác trên các giá trị có kiểu dữ liệu số.

Phép	Ý nghĩa	Kiểu của đối số	Kiểu của

toán			kết quả
-	Phép lấy số đối	Kiểu số (Integer, Single…)	Như kiểu đối số
+	Phép cộng hai số	Kiểu số (Integer, Single…)	Như kiểu đối số
-	Phép trừ hai số	Kiểu số (Integer, Single…)	Như kiểu đối số
*	Phép nhân hai số	Kiểu số (Integer, Single…)	Như kiểu đối số
/	Phép chia hai số	Kiểu số (Integer, Single…)	Single hay Double
\	Phép chia lấy phần nguyên	Integer, Long	Integer, Long
Mod	Phép chia lấy phần dư	Integer, Long	Integer, Long
^	Tính lũy thừa	Kiểu số (Integer, Single…)	Như kiểu đối số

1. Các phép toán quan hệ

Đây là các phép toán mà giá trị trả về của chúng là một giá trị kiểu Boolean (TRUE hay FALSE).

Phép toán	Ý nghĩa
=	So sánh bằng nhau
<>	So sánh khác nhau
>	So sánh lớn hơn
<	So sánh nhỏ hơn
>=	So sánh lớn hơn hoặc bằng
<=	So sánh nhỏ hơn hoặc bằng

1. Các phép toán Logic: là các phép toán tác động trên kiểu Boolean và cho kết quả là kiểu Boolean. Các phép toán này bao gồm AND (và), OR (hoặc), NOT (phủ định). Sau đây là bảng giá trị của các phép toán:

X	Y	X AND Y	X OR Y	NOT X
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

Câu lệnh

Một câu lệnh (statement) xác định một công việc mà chương trình phải thực hiện để xử lý dữ liệu đã được mô tả và khai báo. Các câu lệnh được ngăn cách với nhau bởi ký tự xuống dòng. Ký tự xuống dòng báo hiệu kết thúc một câu lệnh.

Lệnh gán

Cú pháp:

<Tên biến> = <Biểu thức>

Ví dụ:

Giả sử ta có khai báo sau:

Dim TodayTemp As Single, MinAge As Integer

Dim Sales As Single, NewSales As Single, FullName As String

Các lệnh sau gán giá trị cho các biến trên:

TodayTemp = 30.5

MinAge = 18

Sales = 200000

NewSales = Sales * 1.2

Giả sử người dùng cần nhập họ và tên vào ô nhập liệu TextBox có thuộc tính Name là txtName, câu lệnh dưới đây sẽ lưu giá trị của ô nhập liệu vào trong biến FullName:

FullName = txtName.Text

Lưu ý: Kiểu dữ liệu của biểu thức (vế phải của lệnh gán) phải phù hợp với biến ta cần gán trị.

Lệnh rẽ nhánh If

- Một dòng lệnh:

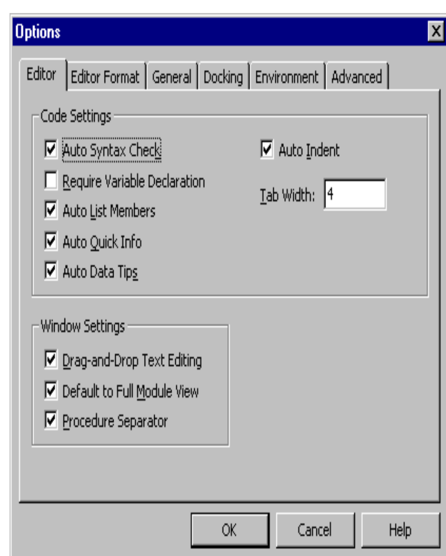
If <điều kiện> Then <dòng lệnh>

- Nhiều dòng lệnh:

If <điều kiện> Then

Các dòng lệnh

End If



Trong đó, <điều kiện>: biểu thức mà kết quả trả về kiểu Boolean.

Ý nghĩa câu lệnh: Các dòng lệnh hay dòng lệnh sẽ được thi hành nếu như điều kiện là đúng. Còn nếu như điều kiện là sai thì câu lệnh tiếp theo sau cấu trúc If ... Then được thi hành.

- Dạng đầy đủ: If ... Then ... Else

If <điều kiện 1> Then

[Khối lệnh 1]

ElseIf <điều kiện 2> Then

[Khối lệnh 2]...

[Else

[Khối lệnh n]]

End If

VB sẽ kiểm tra các điều kiện, nếu điều kiện nào đúng thì khối lệnh tương ứng sẽ được thi hành. Ngược lại nếu không có điều kiện nào đúng thì khối lệnh sau từ khóa Else sẽ được thi hành.

Ví dụ:

If (TheColorYouLike = vbRed) Then

MsgBox "You are a lucky person"

ElseIf (TheColorYouLike = vbGreen) Then

MsgBox "You are a hopeful person"

ElseIf (TheColorYouLike = vbBlue) Then

MsgBox "You are a brave person"

ElseIf (TheColorYouLike = vbMagenta) Then

MsgBox "You are a sad person"

Else

MsgBox "You are an average person"

End If

Lệnh lựa chọn Select Case

Trong trường hợp có quá nhiều các điều kiện cần phải kiểm tra, nếu ta dùng cấu trúc rẽ nhánh If…Then thì đoạn lệnh không được trong sáng, khó kiểm tra, sửa đổi khi có sai sót. Ngược lại với cấu trúc Select…Case, biểu thức điều kiện sẽ được tính toán một lần vào đầu cấu trúc, sau đó VB sẽ so sánh kết quả với từng trường hợp (Case). Nếu bằng nó thi hành khối lệnh trong trường hợp (Case) đó.

Select Case <biểu thức kiểm tra>

Case <Danh sách kết quả biểu thức 1>

[Khối lệnh 1]

Case <Danh sách kết quả biểu thức 2>

[Khối lệnh 2]

.

.

.

[Case Else

[Khối lệnh n]]

End Select

Mỗi danh sách kết quả biểu thức sẽ chứa một hoặc nhiều giá trị. Trong trường hợp có nhiều giá trị thì mỗi giá trị cách nhau bởi dấu phẩy (,). Nếu có nhiều Case cùng thỏa điều kiện thì khối lệnh của Case đầu tiên sẽ được thực hiện.

Ví dụ của lệnh rẽ nhánh If…Then ở trên có thể viết như sau:

```
Select Case TheColorYouLike

Case vbRed

MsgBox "You are a lucky person"

Case vbGreen

MsgBox "You are a hopeful person"

Case vbBlue

MsgBox "You are a brave person"

Case vbMagenta

MsgBox "You are a sad person"

Case Else

MsgBox "You are an average person"

End Select
```

Toán tử Is & To

Toán tử Is: Được dùng để so sánh <Biểu thức kiểm tra> với một biểu thức nào đó.

Toán tử To: Dùng để xác lập miền giá trị của <Biểu thức kiểm tra>.

Ví dụ:

```
Select Case Tuoi

Case Is <18

MsgBox “Vi thanh nien”
```

Case 18 To 30

MsgBox “Ban da truong thanh, lo lap than di”

Case 31 To 60

MsgBox “Ban dang o lua tuoi trung nien”

Case Else

MsgBox “Ban da lon tuoi, nghi huu duoc roi day!”

End Select

Lưu ý: Trong ví dụ trên không thể viết Case Tuoi < 18.

Cấu trúc lặp

Các cấu trúc lặp cho phép thi hành một khối lệnh nào đó nhiều lần.

1. Lặp không biết trước số lần lặp

Khối lệnhDo ... Loop: Đây là cấu trúc lặp không xác định trước số lần lặp, trong đó, số lần lặp sẽ được quyết định bởi một biểu thức điều kiện. Biểu thức điều kiện phải có kết quả là True hoặc False. Cấu trúc này có 4 kiểu:

Kiểu 1:

Do While <điều kiện>

<khối lệnh> Đkiện

Loop

Đúng Sai

Khối lệnh sẽ được thi hành đến khi nào điều kiện không còn đúng nữa. Do biểu thức điều kiện được kiểm tra trước khi thi hành khối lệnh, do đó có thể khối lệnh sẽ không được thực hiện một lần nào cả.

Kiểu 2:

Do

<khối lệnh>

Loop While <điều kiện>

Khối lệnh sẽ được thực hiện, sau đó biểu thức điều kiện được kiểm tra, nếu điều kiện còn đúng thì, khối lệnh sẽ được thực hiện tiếp tục. Do biểu thức điều kiện được kiểm tra sau, do đó khối lệnh sẽ được thực hiện ít nhất một lần.

Kiểu 3:

Do Until <điều kiện>

<khối lệnh>

Loop

Cũng tương tự như cấu trúc Do While ... Loop nhưng khác biệt ở chỗ là khối lệnh sẽ được thi hành khi điều kiện còn sai.

Kiểu 4:

Do

<khối lệnh>

Loop Until <điều kiện>

Khối lệnh được thi hành trong khi điều kiện còn sai và có ít nhất là một lần lặp.

Ví dụ: Đoạn lệnh dưới đây cho phép kiểm tra một số nguyên N có phải là số nguyên tố hay không?

```
Dim i As Integer

i = 2

Do While (i <= Sqr(N)) And (N Mod i = 0)

i = i + 1

Loop

If (i > Sqr(N)) And (N <> 1) Then

MsgBox Str(N) & “ la so nguyen to”

Else

MsgBox Str(N) & “ khong la so nguyen to”

End If
```

Trong đó, hàm Sqr: hàm tính căn bậc hai của một số

Lặp biết trước số lần lặp

- For ... Next

Đây là cấu trúc biết trước số lần lặp, ta dùng biến đếm tăng dần hoặc giảm dần để xác định số lần lặp.

```
For <biến đếm> = <điểm đầu> To <điểm cuối> [Step <bước nhảy>]

[khối lệnh]

Next
```

Biến đếm, điểm đầu, điểm cuối, bước nhảy là những giá trị số (Integer, Single,…). Bước nhảy có thể là âm hoặc dương. Nếu bước nhảy là số âm thì điểm đầu phải lớn hơn điểm cuối, nếu không khối lệnh sẽ không được thi hành.

Khi Step không được chỉ ra, VB sẽ dùng bước nhảy mặc định là một.

Ví dụ: Đoạn lệnh sau đây sẽ hiển thị các kiểu chữ hiện có của máy bạn.

Private Sub Form_Click( )

Dim i As Integer

For i = 0 To Screen.FontCount

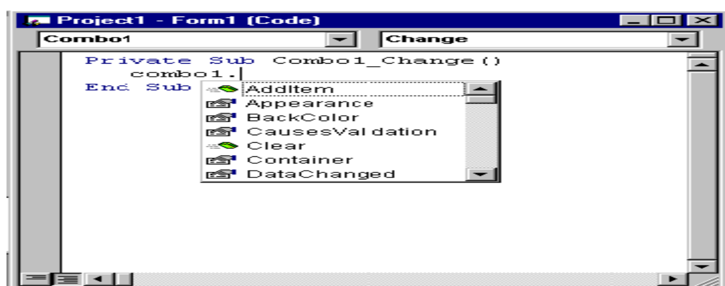
MsgBox Screen.Fonts(I)

Next

End Sub

Ví dụ: Tính N!

- TextBox: Name:txtNum Bước 1: Thiết kế chương trình có giao diện:



Label: Name: lblKQ

- Bước 2: Sự kiện Command1_Click được xử lý:

```
Private Sub Command1_Click()

Dim i As Integer, n As Integer, Kq As Long

n = Val(txtNum.Text)

Kq = 1

For i = 1 To n

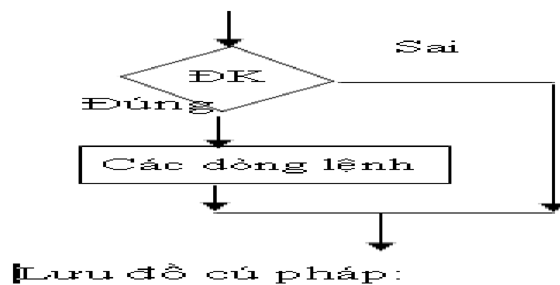
Kq = Kq * i

Next

lblKQ.Caption = Str(Kq)

End Sub
```

- Lưu dự án và chạy chương trình ta được kết quả như hình dưới:



- For Each ... Next

Tương tự vòng lặp For ... Next, nhưng nó lặp khối lệnh theo số phần tử của một tập các đối tượng hay một mảng thay vì theo số lần lặp xác định. Vòng lặp này tiện lợi khi ta không biết chính xác bao nhiêu phần tử trong tập hợp.

For Each <phần tử> In <nhóm>

<khối lệnh>

Next <phần tử>

Lưu ý:

- Phần tử trong tập hợp chỉ có thể là biến Variant, biến Object, hoặc một đối tượng trong Object Browser.

- Phần tử trong mảng chỉ có thể là biến Variant.

- Không dùng For Each ... Next với mảng chứa kiểu tự định nghĩa vì Variant không chứa kiểu tự định nghĩa.

Chương trình con

Khái niệm

Trong những chương trình lớn, có thể có những đoạn chương trình viết lặp đi lặp lại nhiều lần, để tránh rườm rà và mất thời gian khi viết chương trình người ta thường phân chia chương trình thành nhiều module, mỗi module giải quyết một công việc nào đó. Các module như vậy gọi là các chương trình con.

Một tiện lợi khác của việc sử dụng chương trình con là ta có thể dễ dàng kiểm tra xác định tính đúng đắn của nó trước khi ráp nối vào chương trình chính và do đó việc xác định sai sót để tiến hành hiệu đính trong chương trình chính sẽ thuận lợi hơn.

Trong Visual Basic, chương trình con có hai dạng là hàm (Function) và thủ tục (Sub).

Hàm khác thủ tục ở chỗ hàm trả về cho lệnh gọi một giá trị thông qua tên của nó còn thủ tục thì không. Do vậy ta chỉ dùng hàm khi và chỉ khi thoả mãn đồng thời các yêu cầu sau đây:

- Ta muốn nhận lại một kết quả (chỉ một mà thôi) khi gọi chương trình con.

- Ta cần dùng tên chương trình con (có chứa kết quả) để viết trong các biểu thức.

Nếu không thỏa mãn hai điều kiện ấy thì dùng thủ tục.

Thủ tục

1. Khái niệm:

Thủ tục là một chương trình con thực hiện một hay một số tác vụ nào đó. Thủ tục có thể có hay không có tham số.

1. Khai báo thủ tục

[Private | Public] [Static] Sub <tên thủ tục> [(<tham số>[As <Kiểu tham số>])]

<Các dòng lệnh> hay <Các khai báo>

End Sub

Trong đó:

- <Tên thủ tục>: Đây là một tên được đặt giống quy tắc tên biến, hằng,…

- <tham số>[: <Kiểu tham số>]: có thể có hay không? Nếu có nhiều tham số thì mỗi tham số phân cách nhau dấu phẩy. Nếu không xác định kiểu tham số thì tham số có kiểu Variant.

Để gọi thủ tục để thực thi, ta có 2 cách:

- <Tên thủ tục> [<Các tham số thực tế>]
- Call <Tên thủ tục> ([<Các tham số thực tế>])

Ví dụ: Thiết kế chương trình kiểm tra xem số nguyên N có phải là số nguyên tố hay không?

- Bước 1: Thiết kế chương trình có giao diện

[missing_resource: .png]

TextBox: Name:txtNum

- Bước 2: Viết thủ tục KtraNgTo trong phần mã lệnh của Form

```
Sub KTraNgTo(N As Integer)

Dim i As Integer

i = 2

Do While (i <= Sqr(N)) And (N Mod i <> 0)

i = i + 1

Loop

If (i > Sqr(N)) And (N <> 1) Then

MsgBox Str(N) & " la so nguyen to"

Else

MsgBox Str(N) & " khong la so nguyen to"

End If

End Sub
```

- Bước 3: Xử lý sự kiện Command1_Click; trong thủ tục xử lý sự kiện này ta có gọi thủ tục KtraNgTo như sau:

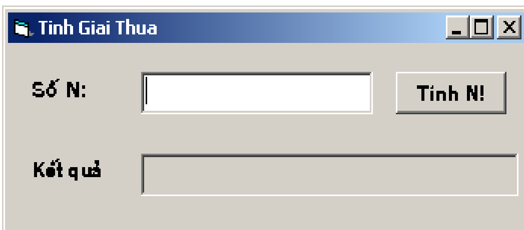
Private Sub Command1_Click()

KTraNgTo Val(txtNum.Text)

‘ Call KtraNgTo(Val(txtNum.Text))

End Sub

- Bước 4: Lưu dự án và chạy chương trình. Ta được kết quả sau:



Trong ví dụ trên thay vì gọi thủ tục bằng lời gọi:

KTraNgTo Val(txtNum.Text)

Ta có thể sử dụng cách khác:

Call KtraNgTo(Val(txtNum.Text))

Hàm

1. Khái niệm

Hàm (Function) là một chương trình con có nhiệm vụ tính toán và cho ta một kết quả. Kết quả này được trả về trong tên hàm cho lời gọi nó.

1. Khai báo hàm

[Private | Public | Static] Function <Tên hàm> [(<tham số>[As <Kiểu tham số>])] _

[As <KIẾU DỮ LIỆU>]

<Các dòng lệnh> hay <Các khai báo>

End Function

Trong đó:

- <Tên hàm>: Đây là một tên được đặt giống quy tắc tên biến, hằng,…

- <tham số>[: <Kiểu tham số>]: có thể có hay không? Nếu có nhiều tham số thì mỗi tham số phân cách nhau dấu phẩy. Nếu không xác định kiểu tham số thì tham số có kiểu Variant.

- <KIỂU DỮ LIỆU>: Kết quả trả về của hàm, trong trường hợp không khai báo As <kiểu dữ liệu>, mặc định, VB hiểu kiểu trả về kiểu Variant.

Khi gọi hàm để thực thi ta nhận được một kết quả. Cần chú ý khi gọi hàm thực thi ta nhận được một kết quả có kiểu chính là kiểu trả về của hàm (hay là kiểu Variant nếu ta không chỉ rõ kiểu trả về trong định nghĩa hàm). Do đó lời gọi hàm phải là thành phần của một biểu thức.

Cú pháp gọi hàm thực thi: <Tên hàm>[(tham số)].

Ví dụ: Tính N!

- TextBox: Name:txtNum Bước 1: Thiết kế chương trình có giao diện:

Label: Name: lblKQ

- Bước 2: Thêm một hàm vào cửa sổ mã lệnh của Form

Function Giaithua(N As Integer) As Long

Dim i As Integer, Kq As Long

Kq = 1

For i = 1 To n

Kq = Kq * i

Next

Giaithua = Kq

End Function

Private Sub Command1_Click()

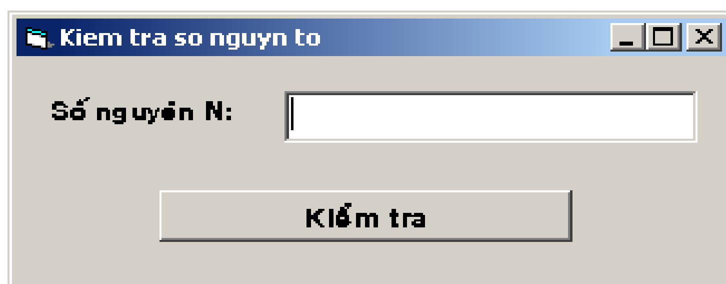
Dim n As Integer

n = Val(txtNum.Text)

lblKQ.Caption = Str(Giaithua(n))

End Sub

Lưu dự án và chạy chương trình ta được kết quả như hình dưới:



Lưu ý: Do khi gọi hàm ta nhận được một kết quả nên bên trong phần định nghĩa hàm, trước khi kết thúc ta phải gán kết quả trả về của hàm thông qua tên hàm (trong ví dụ trên là dòng lệnh Giaithua = Kq)

Truy xuất dữ liệu trong Visual Basic

Các khái niệm

- Module:

- Một ứng dụng đơn giản có thể chỉ có một biểu mẫu, lúc đó tất cả mã lệnh của ứng dụng đó được đặt trong cửa sổ mã lệnh của biểu mẫu đó (gọi là Form Module). Khi ứng dụng được phát triển lớn lên, chúng ta có thể có thêm một số biểu mẫu nữa và lúc này khả năng lặp đi lặp lại nhiều lần của một đoạn mã lệnh trong nhiều biểu mẫu khác nhau là rất lớn.

- Để tránh việc lặp đi lặp lại trên, ta tạo ra một Module riêng rẽ chứa các chương trình con được dùng chung. Visual Basic cho phép 3 loại Module:

Module biểu mẫu (Form module): đi kèm với mỗi một biểu mẫu là một module của biểu mẫu đó để chứa mã lệnh của biểu mẫu này. Với mỗi điều khiển trên biểu mẫu, module biểu mẫu chứa các chương trình con và chúng sẵn sàng được thực thi để đáp ứng lại các sự kiện mà người sử dụng ứng dụng tác động trên điều khiển. Module biểu mẫu được lưu trong máy tính dưới dạng các tập tin có đuôi là *.frm.

Module chuẩn (Standard module): Mã lệnh không thuộc về bất cứ một biểu mẫu hay một điều khiển nào sẽ được đặt trong một module đặc biệt gọi là module chuẩn (được lưu với đuôi *.bas). Các chương trình con được lặp đi lặp lại để đáp ứng các sự kiện khác nhau của các điều khiển khác nhau thường được đặt trong module chuẩn.

Module lớp (Class module): được sử dụng để tạo các điều khiển được gọi thực thi trong một ứng dụng cụ thể. Một module chuẩn chỉ chứa mã

lệnh nhưng module lớp chứa cả mã lệnh và dữ liệu, chúng có thể được coi là các điều khiển do người lập trình tạo ra (được lưu với đuôi *.cls).

- Phạm vi (scope): xác định số lượng chương trình có thể truy xuất một biến. Một biến sẽ thuộc một trong 3 loại phạm vi:

Phạm vi biến cục bộ.

Phạm vi biến module.

Phạm vi biến toàn cục.

Biến toàn cục

- Khái niệm: Biến toàn cục là biến có phạm vi hoạt động trong toàn bộ ứng dụng.
- Khai báo:

Global <Tên biến> [As <Kiểu dữ liệu>]

Biến cục bộ

- Khái niệm: Biến cục bộ là biến chỉ có hiệu lực trong những chương trình mà chúng được định nghĩa.
- Khai báo:

Dim <Tên biến> [As <Kiểu dữ liệu>]

- Lưu ý:

Biến cục bộ được định nghĩa bằng từ khóa Dim sẽ kết thúc ngay khi việc thi hành thủ tục kết thúc.

Biến Module

- Khái niệm: Biến Module là biến được định nghĩa trong phần khai báo (General|Declaration) của Module và mặc nhiên phạm vi hoạt động của nó là toàn bộ Module ấy.
- Khai báo:

- Biến Module được khai báo bằng từ khóa Dim hay Private & đặt trong phần khai báo của Module.

Ví dụ:

Private Num As Integer

- Tuy nhiên, các biến Module này có thể được sử dụng bởi các chương trình con trong các Module khác. Muốn thế chúng phải được khai báo là Public trong phân Khai báo (General|Declaration) của Module.

Ví dụ:

Public Num As Integer

Lưu ý: Không thể khai báo biến với từ khóa là Public trong chương trình con.

Truyền tham số cho chương trình con

- Khái niệm

Một chương trình con đôi lúc cần thêm một vài thông tin về trạng thái của đoạn mã lệnh mà nó định nghĩa để thực thi. Những thông tin này là các biến được truyền vào khi gọi chương trình con, các biến này gọi là tham số của chương trình con.

Có hai cách để truyền tham số cho chương trình con: Truyền bằng giá trị & truyền bằng địa chỉ.

- Truyền tham số bằng giá trị

Với cách truyền tham số theo cách này, mỗi khi một tham số được truyền vào, một bản sao của biến đó được tạo ra. Nếu chương trình con có thay đổi giá trị, những thay đổi này chỉ tác động lên bản sao của biến. Trong VB, từ khóa ByVal được dùng để xác định tham số được truyền bằng giá trị.

Ví dụ:

```
Sub Twice (ByVal Num As Integer)

Num = Num * 2

Print Num

End Sub

Private Sub Form_Click()

Dim A As Integer

A = 4

Print A

Twice A

Print A

End Sub
```

Kết quả thực hiện của đoạn chương trình trên:

4

8

4

- Truyền tham số bằng địa chỉ

Truyền tham số theo địa chỉ cho phép chương trình con truy cập vào giá trị gốc của biến trong bộ nhớ. Vì thế, giá trị của biến có thể sẽ bị thay đổi bởi đoạn mã lệnh trong chương trình con. Mặc nhiên, trong VB6 các tham số được truyền theo địa chỉ; tuy nhiên ta có thể chỉ định một cách tường minh nhờ vào từ khóa ByRef.

Ví dụ:

```
Sub Twice (Num As Integer)

Num = Num * 2

Print Num

End Sub

Private Sub Form_Click()

Dim A As Integer

A = 4

Print A

Twice A

Print A

End Sub
```

Kết quả thực hiện của đoạn chương trình trên:

4

8

8

Bẫy lỗi trong Visual Basic

Các thao tác bẫy các lỗi thực thi của chương trình là cần thiết đối với các ngôn ngữ lập trình. Người lập trình khó kiểm soát hết các tình huống có thể gây ra lỗi. Chẳng hạn người ta khó có thể kiểm tra chặt chẽ việc người dùng đang chép dữ liệu từ đĩa mềm (hay CD) khi chúng không có trong ổ đĩa. Nếu có các thao tác bẫy lỗi ở đây thì tiện cho người lập trình rất nhiều.

Visual Basic cũng cung cấp cho ta một số cấu trúc để bẫy các lỗi đang thực thi.

Cú pháp:

Dạng 1:

On Error GoTo <Tên nhãn>

<Các câu lệnh có thể gây ra lỗi>

<Tên nhãn>:

<Các câu lệnh xử lý lỗi>

Ý nghĩa:

- <Tên nhãn>: là một tên được đặt theo quy tắc của một danh biểu.

- Nếu một lệnh trong <Các câu lệnh có thể gây ra lỗi> thì khi chương trình thực thi đến câu lệnh đó, chương trình sẽ tự động nhảy đến đoạn chương trình định nghĩa bên dưới <Tên nhãn> để thực thi.

Dạng 2:

On Error Resume Next

<Các câu lệnh có thể gây ra lỗi>

Ý nghĩa:

- Nếu một lệnh trong <Các câu lệnh có thể gây ra lỗi> thì khi chương trình thực thi đến câu lệnh đó, chương trình sẽ tự động bỏ qua câu lệnh bị lỗi và thực thi câu lệnh kế tiếp.

Giới thiệu học phần Lập Trình Hệ Thống

Chương 1 TỔ CHỨC BỘ XỬ LÝ INTEL-8086

Mục đích:

- Cấu trúc bên trong CPU Intel-8086

- Tập thanh ghi

- Tổ chức bộ nhớ và dữ liệu

- Khái quát các bộ xử lý Intel khác như: 80386, 80486, Pentium

1.1. BỘ XỬ LÝ INTEL-8086 (CPU-8086)

1.1.1. Cấu trúc tổng quát

Intel-8086 là một CPU 16 bit (bus dữ liệu ngoại có 16 dây). Nó được dùng để chế tạo các máy vi tính PC-AT đầu tiên của hãng IBM vào năm 1981. Trong thực tế, hãng IBM đã dùng CPU 8088 (là một dạng của CPU 8086 với bus số liệu giao tiếp với ngoại vi là 8 bit) để chế tạo máy vi tính cá nhân đầu tiên gọi là PC-XT.

Cho đến nay CPU 8086 đã không ngừng cải tiến và đã trải qua các phiên bản 80186, 80286, 80386, 80486, Pentium (80586), Pentium Pro, Pentium MMX, Pentium II, III, 4. Các CPU trên tương thích từ trên xuống (downward compatible) nghĩa là tập lệnh của các CPU mới chế tạo gồm các tập lệnh của CPU chế tạo trước đó được bổ sung thêm nhiều lệnh mạnh khác.

Cấu trúc tổng quát của CPU-8086 có dạng như hình 1.1, gồm 2 bộ phận chính là: Bộ thực hiện lệnh và bộ phận giao tiếp bus.

1. Bộ phận thực hiện lệnh (EU):

Thi hành các tác vụ mà lệnh yêu cầu như: Kiểm soát các thanh ghi (đọc/ghi), giải mã và thi hành lệnh. Trong EU có bộ tính toán và luận lý (ALU) thực hiện được các phép toán số học và luận lý. Các thanh ghi đa dụng là các ô nhớ bên trong CPU chứa dữ liệu tương tự như ô nhớ trong bộ nhớ. Cờ cũng là một thanh ghi dùng để ghi lại trạng thái hoạt động của ALU. Thanh ghi lệnh chứa nội dung lệnh hiện tại mà CPU đang thực hiện.

Các thanh ghi và bus trong EU đều là 16 bit. EU không kết nối trực tiếp với bus hệ thống bên ngoài. Nó lấy lệnh từ hàng chờ lệnh mà BIU cung cấp. Khi có yêu cầu truy xuất bộ nhớ hay ngoại vi thì EU yêu cầu BIU làm việc. BIU có thể tái định địa chỉ để cho phép EU truy xuất đầy đủ 1 MB (8086 có 20 đường địa chỉ ngoại).

2. Bộ giao tiếp bus (BIU):

BIU thực hiện chức năng giao tiếp giữa EU với bên ngoài (Bộ nhớ, thiết bị ngoại vi …) thông qua hệ thống BUS ngoại (bus dữ liệu và bus địa chỉ). BIU thực hiện tất cả các tác vụ về bus mỗi khi EU có yêu cầu. Khi EU cần trao đổi dữ liệu với bên ngoài, BIU sẽ tính toán địa chỉ và truy xuất dữ liệu để phục vụ theo yêu cầu EU.

Trong BIU có 5 thanh ghi CS, DS, ES, SS và IP chứa địa chỉ. Thanh ghi IP chứa địa chỉ của lệnh sẽ được thi hành kế tiếp nên gọi là con trỏ lệnh.

EU và BIU liên lạc với nhau thông qua hệ thống bus nội. Trong khi EU đang thực hiện lệnh thì BIU lấy lệnh từ bộ nhớ trong nạp đầy vào hàng chờ lệnh (6 bytes). Do đó EU không phải đợi lấy lệnh từ bộ nhớ. Đây là một

dạng đơn giản của cache để tăng tốc độ đọc lệnh.

[missing_resource: graphics1.wmf]

Hình 1.1: Sơ đồ khối của CPU 8086

1.1.2. Các thanh ghi của 8086

Thanh ghi (register) là thành phần lưu trữ dữ liệu bên trong CPU, mỗi thanh ghi có độ dài nhất định (16 bit hoặc 8 bit) và được nhận biết bằng một tên riêng. Tùy vào độ dài và chức năng mà thanh ghi có công dụng chứa dữ liệu hoặc kết quả của phép toán, hoặc là các địa chỉ dùng để định vị bộ nhớ khi cần thiết.

Nội dung của thanh ghi được truy xuất thông qua tên riêng của nó, do đó tên thanh ghi là từ khóa quan trọng cần phải lưu ý trong lập trình.

CPU-8086 có 16 thanh ghi, mỗi thanh ghi là 16 bit, có thể chia 4 nhóm sau:

1. Thanh ghi đoạn: Gồm 4 thanh ghi 16 bit: CS, DS, ES, SS. Đây là những thanh ghi dùng để chứa địa chỉ đoạn của các ô nhớ khi cần truy xuất. Mỗi thanh ghi đoạn quản lý 1 đoạn tối đa 64K ô nhớ trong bộ nhớ trong. Người sử dụng chỉ được phép truy xuất ô nhớ dựa vào địa chỉ tương đối. CPU (cụ thể là BIU) có nhiệm vụ chuyển đổi địa chỉ tương đối thành địa chỉ tuyệt đối để truy xuất vào ô nhớ tuyệt đối tương ứng trong bộ nhớ. (Xem phần tổ chức bộ nhớ)

CS: Thanh ghi đoạn mã lệnh, lưu địa chỉ đoạn chứa mã lệnh chương trình của người sử dụng

DS: Thanh ghi đoạn dữ liệu, lưu địa chỉ đoạn chứa dữ liệu (các biến) trong chương trình.

ES: Thanh ghi đoạn dữ liệu thêm, lưu địa chỉ đoạn chứa dữ liệu thêm trong chương trình.

SS: Thanh ghi đoạn ngăn xếp, lưu địa chỉ đoạn của vùng ngăn xếp.

15 0	
CS	Code Segment
DS	Data Segment
ES	Extra data Segment
SS	Stack Segment

Thông thường bốn thanh ghi này có thể chứa những giá trị khác nhau, do đó chương trình có thể được truy cập trên bốn đoạn khác nhau và chương trình chỉ có thể truy cập cùng 1 lúc tối đa bốn đoạn. Mặc khác, đối với những chương trình nhỏ, chỉ sử dụng 1 đoạn duy nhất, khi đó cả bốn thanh ghi đều chứa cùng giá trị địa chỉ đoạn, gọi là đoạn chung.

1. Thanh ghi đa dụng (General Register): Bao gồm bốn thanh ghi đa dụng 16 bit (AX, BX, CX, DX). Mỗi thanh ghi đa dụng có thể được sử dụng với nhiều mục đích khác nhau, tuy nhiên từng thanh ghi có công

dụng riêng của nó.

15 8	7 0	
AH	AL	AX (Accumulator)
BH	BL	BX (Base register)
CH	CL	CX (Count register)
DH	DL	DX (Data register)

AX : Là thanh ghi tích lũy cơ bản. Mọi tác vụ vào/ra đều dùng thanh ghi này, tác vụ dùng số liệu tức thời, một số tác vụ chuỗi ký tự và các lệnh tính toán đều dùng thanh AX.

BX: Thanh ghi nền dùng để tính toán địa chỉ ô nhớ.

CX: Là thanh ghi đếm, thường dùng để đếm số lần trong một lệnh vòng lặp hoặc lệnh xử lý chuỗi ký tự.

DX: Thanh ghi dữ liệu, thường chứa địa chỉ của một số lệnh vào/ra, lệnh tính toán số học (kể cả lệnh nhân và chia).

Mỗi thanh ghi 16 bit có thể chia đôi thành 2 thanh ghi 8 bit. Do đó, CPU-8086 có 8 thanh ghi 8 bit là: AH, AL; BH, BL; CH, CL; DH, DL (thanh ghi AH và AL tương ứng với byte cao và byte thấp của thanh ghi AX, tương tự cho các thanh ghi 8 bit còn lại).

Ví dụ: AX = 1234h => AH = 12h, AL = 34h

1. Thanh ghi con trỏ và chỉ số (Pointer & Index register): Chức năng chung của nhóm thanh ghi này là chứa địa chỉ độ dời của ô nhớ trong vùng dữ liệu hay ngăn xếp.

SI : Thanh ghi chỉ số nguồn

DI : Thanh ghi chỉ số đích

BP: Thanh ghi con trỏ nền dùng để lấy số liệu từ ngăn xếp.

SP : Thanh ghi con trỏ ngăn xếp luôn chỉ vào đỉnh ngăn xếp.

15 0	
SI	Source Index Reg.
DI	Destination Index Reg.
BP	Base Pointer Reg.

SP	Stack Pointer Reg.

SI và DI chứa địa chỉ độ dời của ô nhớ tương ứng trong đoạn có địa chỉ chứa trong DS hoặc ES (dữ liệu, còn gọi là Biến). Còn BP và SP chứa địa chỉ độ dời của ô nhớ tương ứng trong đoạn có địa chỉ chứa trong SS, dùng để thâm nhập số liệu trong ngăn xếp.

1. Thanh ghi Đếm chương trình và thanh ghi trạng thái (Cờ):

15 0	
F	Flag Register.
IP	Intrucstion Pointer Reg.

- Thanh ghi con trỏ lệnh IP (còn gọi là PC – đếm chương trình) là thanh ghi 16 bit chứa địa chỉ của lệnh kế tiếp mà CPU sẽ thực hiện trong. Các lệnh của chương trình có địa chỉ đoạn trong CS.
- Thanh ghi Cờ (F) dài 16 bit, mỗi bit là một cờ. Mỗi cờ có giá trị 1 (gọi là SET –Đặt) hoặc 0 (gọi là CLEAR – Xóa). Hình 1.2 mô tả 9 bit trong số 16 bit tương ứng với 9 cờ trạng thái (các bit còn lại dùng cho dự trữ mở rộng khi thiết kế các CPU khác)

Thanh ghi cờ được chia thành hai nhóm:

- Nhóm cờ điều khiển (bảng 1.1) bao gồm các cờ dùng để điều khiển sự hoạt động của CPU và giá trị của cờ được thiết lập bằng các lệnh phần mềm.
- Nhóm cờ trạng thái (bảng 1.2) bao gồm các cờ phản ánh kết quả thực hiện lệnh cũng như trạng thái của CPU

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				O	D	I	T	S	Z		A		P		C

Dự trữ

Hình 1.2: Cấu trúc thanh ghi Cờ

KÝ HIỆU	TÊN	Ý NGHĨA
Nhóm cờ điều khiển		

TF	Bẩy (Trap)	TF = 1: cho phép chương trình chạy từng bước
IF	Ngắt (Interrupt)	IF = 1: cho phép ngắt phần cứng
DF	Hướng (Direction)	DF = 1: thì SI và DI giảm 1 cho mỗi vòng lặp
Nhóm cờ trạng thái		
CF	Số giữ(Carry)	CF = 1: khi có số nhớ hoặc mượn từ MSB trong phép cộng hoặc trừ.(Có tthể bị thay đổi theo lệnh ghi dịch và quay)
PF	Chẩn lẻ (Parity)	PF = 1: khi byte thấp của thanh ghi kết quả một phép tính có số lượng bit 1 chẳn
AF	Số giữ phụ (Hafl)	AF = 1: khi có nhớ hoặc mượn từ bit 3 trong phép cộng hoặc trừ. Dùng trong các lệnh với số BCD
ZF	Zero	ZF = 1: khi kết quả của một phép tính bằng 0
SF	Dấu (Sign)	SF = 1 khi kết quả phép tính là âm (MSB=1).SF = 0 khi kết quả dương (MSB=0)
OF	Tràn (Overflow)	OF = 1: nếu kết quả vượt quá khả năng tính toán của CPU

Bảng 1.1: Ý nghĩa cờ

1.1.3. Trạng thái tràn:

Trạng thái tràn có thể không xảy ra (nếu không tràn) hoặc xảy ra (nếu tràn có dấu, tràn không dấu, đồng thời tràn có dấu và không dấu). Nói chung là có 2 trạng thái tràn: Tràn không dấu và Tràn có dấu.

Lưu ý: Nếu một giá trị có MSB=1 (bit dấu) thì CPU luôn luôn cho đó là số có dấu.

a. Tràn không dấu: CF=1

Ví dụ: ADD AX, BX ; với AX = 0FFFFh, BX = 1

- Nếu xem đây là các số không dấu thì AX không đủ chứa kết quả nên TRÀN không dấu, vậy CF = 1
- Nếu xem đây là các số có dấu thì kết quả sẽ là 0 (vì AX = -1) nên không tràn, do đó OF = 0

b. Tràn có dấu: OF = 1

Ví dụ: ADD AX, BX ; với AX = BX = 7FFFh = 32767

- Nếu xem đây là các số không dấu thì AX = 7FFFh + 7FFFh = 0FFFEh = 65534 nên không tràn.
- Nếu xem đây là các số có dấu thì tràn vì kết quả vượt quá phạm vi cho phép đối với số có dấu (cộng 2 số dương, kết quả lại là số âm). Thật sự trong trường hợp này, CPU sẽ làm cho OF = 1 theo qui tắc "Nhớ ra và vào MSB xảy ra không đồng thời” nghĩa là có nhớ vào MSB nhưng không có nhớ ra hoặc ngược lại thì tràn và không có hoặc có nhớ ra và vào MSB thì không tràn.

1.2. BỘ NHỚ TRONG CỦA INTEL-80x86

1.2.1. Tổ chức dữ liệu

Bộ nhớ trong được tổ chức thành mảng gồm các ô nhớ 8 bit liên tục nhau. Các dữ liệu có thể được ghi vào hoặc đọc ra (gọi là truy xuất) từ bất cứ vị trí ô nhớ nào. Mỗi ô nhớ 8 bit được phần cứng quản lý bằng một địa chỉ vật lý duy nhất. Việc truy xuất nội dung ô nhớ phải bằng địa chỉ vật lý này.

Dữ liệu 8 bit được lưu trữ bằng một ô nhớ và địa chỉ của ô nhớ chính là địa chỉ dùng để truy xuất dữ liệu. Dữ liệu nhiều hơn 8 bit được lưu trữ bởi nhiều ô nhớ liên tục nhau. Theo quy ước Intel, byte dữ liệu cao được lưu ở ô nhớ có địa chỉ cao và byte dữ liệu thấp hơn lưu ở ô nhớ có địa chỉ thấp hơn. Khi đó, địa chỉ dùng để truy xuất dữ liệu là địa chỉ của ô nhớ thấp (ô nhớ chứa byte thấp nhất của dữ liệu)

Hình 1.3 mô tả việc tổ chức các dữ liệu có độ dài khác nhau trong bộ nhớ. Giá trị 5Fh (1 byte) được lưu trữ ở địa chỉ 0010h. Giá trị 0A0B1h (2 byte) được lưu trữ bởi 2 ô nhớ có địa chỉ 0015h và 0016h, địa chỉ để truy xuất giá trị này là 0015h. Còn giá trị 0A2B1C0h (3 byte) được lưu trữ bởi 3 ô nhớ 0012h, 0013h và 0014h, do đó địa chỉ truy xuất giá trị ấy là 0012h.

	Bộ nhớ	Địa chỉ
(A0h là byte cao, B1h là byte thấp)	A0h	0016h
Giá trị: 0A0B1h	B1h	0015h
	A2h	0014h
(C0h byte thấp nhất, A2h byte cao nhất)	B1h	0013h
Giá trị: 0A2B1C0h	C0h	0012h
		0011h
Giá trị: 5Fh	5Fh	0010h

Hình 1.3: Tổ chức dữ liệu trong bộ nhớ

1.2.2. Sự phân đoạn bộ nhớ trong

CPU 8086 có không gian địa chỉ là 1MB (ứng với 20 bit địa chỉ) Vậy CPU 8086 có thể quản lý bộ nhớ trong là 220 = 1MB. Bộ nhớ 1 MB này được CPU-8086 quản lý bằng nhiều đoạn 64 KB. Các đoạn có thể tách rời hoặc có thể chồng lên nhau.

Mỗi đoạn có một địa chỉ đoạn 16 bit duy nhất, tùy vào mục đích sử dụng đoạn mà địa chỉ đoạn được lưu trữ trong thanh ghi đoạn tương ứng.

Đối với người lập trình, Địa chỉ của ô nhớ trong bộ nhớ được xác định bởi hai thông số 16 bit (gọi là địa chỉ logic): Địa chỉ Đoạn (segment) và địa chỉ độ dời (offset).

Cách viết: Segment : Offset

Địa chỉ vật lý của ô nhớ khi truy xuất sẽ được BIU tự động chuyển đổi từ địa chỉ logic bằng cách dịch trái thanh ghi đoạn bốn bit (tức nhân nội dung của thanh ghi đoạn cho 16) rồi cộng với địa chỉ độ dời. Vì vậy, người lập trình không cần địa chỉ vật lý của ô nhớ mà chỉ cần biết địa chỉ logic của ô nhớ.

Ví dụ: đoạn CS có giá trị là 1002h, địa chỉ độ dời của ô nhớ K trong đoạn CS là 500h (CS:0500h hoặc 1002h:0500h). Khi đó, địa chỉ vật lý của ô nhớ K được tính như sau:

10020h (dịch trái địa chỉ đoạn 4 bit)

+ 0500h (độ dời)

10520h (địa chỉ vật lý)

Trong ví dụ trên, đoạn CS có điểm bắt đầu ở địa chỉ vật lý 10020h. Độ dời 500h là khoảng cách từ địa chỉ của điểm bắt đầu của đoạn CS đến ô nhớ K (xem hình 1.4)

	Bộ nhớ	Địa chỉ vật lý	
		10521h	
Ô nhớ K		10520h	Độ dời 500h tính từ điểm đầu đoạn CS
trong đoạn CS (CS:500h)		1051Fh	
	- - -	- - -	
	- - -	- - -	
Ô nhớ có độ dời 05h		10025h	501h ô nhớ
		10024h	
		10023h	
Ô nhớ có độ dời 01h		10022h	
Ô nhớ có độ dời 0h trong đoạn CS (CS:00h)			

		10021h	
		10020h	Điểm đầu đoạn CS
		1001Fh	

Hình 1.4: Địa chỉ vật lý của ô nhớ K trong đoạn CS

Khi truy xuất ô nhớ, BIU lấy sẽ sử dụng địa chỉ đoạn trong thanh ghi đoạn tương ứng với tính chất của ô nhớ cần truy xuất:

- Ô nhớ là Code (Mã lệnh) thì đoạn tương ứng là CS.

- Ô nhớ là Data (dữ liệu) thì đoạn tương ứng là DS.

- Ô nhớ nằm trên ngăn xếp thì dùng đoạn SS.

- Khi truy xuất chuỗi, DI và SI luôn chứa độ dời của ô nhớ trong đoạn DS hay ES

Khi khởi động, CPU 8086 nhảy đến địa chỉ vật lý cao nhất của bộ nhớ trong (đoạn CS = 0FFFFh và độ dời 0) để lấy lệnh. Địa chỉ này ứng với địa chỉ của ROM-BIOS của bộ nhớ trong nơi chứa chương trình khởi động máy tính.

1.3. Địa chỉ các ngoại vi

Các ngoại vi đều có địa chỉ riêng từ 0 đến 64K. CPU 8086 dùng các lệnh riêng biệt để truy xuất ngoại vi và bộ nhớ trong. Muốn truy xuất ngoại vi, BIU chỉ cần đưa địa chỉ của ngoại vi lên 16 bit thấp của bus địa chỉ (không có đoạn).

I/O64 kByteMemory 1 MByteIntel-8086 ***SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.***

Hình 1.5: Cấu trúc đơn giản của máy tính

1.4. CÁC BỘ XỬ LÝ INTEL KHÁC

Ngoài CPU-8086, Intel đã cho ra đời thế hệ CPU mới hơn, nhiều tính năng hơn và mạnh hơn như: 80186, 80286, 80386, 80486, Celeron và Pentium (80586). Ngày nay, sức mạnh và tính năng của CPU Intel tăng vượt bậc nhờ công nghệ mới như: Centrino, Hyper Threading, Core Duo.

Bắt đầu từ CPU 80286, hãng Intel đã đưa vào một số cải tiến có ý nghĩa như tăng bus địa chỉ lên 24 bit và có thể vận hành với chế độ bảo vệ (protected mode). Chế độ này cho phép CPU 80286 vận hành trong một hệ điều hành đa nhiệm (multitasking).

1.4.1. Bộ xử lý 80386

Hãng Intel đã thành công lớn khi chế tạo CPU 80386. Đây vẫn còn là một CPU CISC thuần túy có bus địa chỉ 32 bit và bus số liệu 32 bit (ta gọi CPU 80386 là một CPU 32 bit).

Các thanh ghi của CPU 80386 đều là thanh ghi 32 bit (Hình 1.5). Một số thanh ghi đa dụng có thể chia thành thanh ghi 16 bit hoặc chia thành thanh ghi 8 bit để đảm bảo tính tương thích với các CPU chế tạo trước đó.

Với bus địa chỉ 32 bit, không gian địa chỉ của CPU 80386 là 4 GB tức 4096 MB. CPU 80386 có 64K cửa vào/ra 8 bit, 16 bit và 32 bit.

CPU 80386 cũng có thể hoạt động với bộ đồng xử lý toán học 80387 (math coprocessor). Bộ đồng xử lý toán học dùng xử lý các phép tính trên các số có dấu chấm động (số lẻ).

I.4.2. Tập thanh ghi của bộ xử lý 80386:

Hình 1.6 mô tả đầy đủ tập thanh ghi của CPU-80386.

Các thanh ghi đa dụng và thanh ghi con trỏ được mở rộng thành thanh ghi 32 bit được gọi tên là: EAX, EBX, ESP, EDI... . Tuy nhiên ta vẫn có thể sử dụng các thanh ghi 16 bit (AX, BX, ...) hoặc 8 bit (AH, AL, BH, BL ...) giống như các thanh ghi 16 bit hoặc 8 bit của bộ xử lý 8086.

Chiều dài các thanh ghi đoạn vẫn giữ nguyên 16 bit nhưng có thêm hai thanh ghi đoạn thêm là FS và GS. Các thanh ghi FS và GS được dùng giống như thanh ghi ES. Nghĩa là CPU-80386 quản lý được bốn đoạn dữ liệu.

Thanh ghi trạng thái SR (Status register) và thanh ghi đếm chương trình PC (program counter) cũng được nâng lên 32 bit. Ngoài các bit trạng thái đã thấy trong thanh ghi trạng thái của CPU-8086 (C, Z, S, ...) thanh ghi trạng thái của CPU-80386 còn có thêm các bit trạng thái như sau:

- IOP (Input/Output protection: bảo vệ vào/ra): Đây là hai bit trạng thái dùng trong chế độ bảo vệ để xác định mức ưu tiên mà một tiến trình phải có để có thể thâm nhập một vùng vào ra. Chỉ hệ điều hành mới có quyền dùng các bit này.
- N (Nested task: tiến trình lồng vào nhau): Trong chế độ bảo vệ, các hệ điều hành dùng bit này để biết có nhiều tiến trình đang vận hành và ít nhất có một tiến trình đang bị gián đoạn.
- R (Resume: tải trục): Bit này cho phép một tiến trình được tiếp tục vận hành lại sau khi bị gián đoạn.
- V (Virtual 8086 mode: chế độ 8086 ảo): Bit này cho phép 80386 đang vận hành ở chế độ bảo vệ, chuyển sang chế độ 8086 ảo.

31 16 15 8 7 0			
EAX		AH	AL
EBX		BH	BL
ECX		CH	CL
EDX		DH	DL
ESP		SP	
EBP		BP	
ESI		SI	
EDI		DI	
ESR		SR	

EPC		PC
CS		
DS		
SS		
ES		
FS		
GS		

Hình 1.6a: Thanh ghi đa dụng và thanh ghi con trỏ

15 0 31 0 19 0			
TR			
LDTR			
	IDTR		
	GDTR		

Hình 1.6b: Thanh ghi quản lý bộ nhớ

31 16 15 0 31 16 15 0					
CR3			DR7		
CR2			DR6		
CR1			DR5		
CR0			DR4		
Hình 1.6c: Thanh ghi điều khiển			DR3		
31 16 15 0			DR2		
TR7			DR1		
TR6			DR0		

Hình 1.6d: Thanh ghi kiểm tra	Hình 1.6e: Thanh ghi gỡ rối

Hình 1.6: Các thanh ghi của CPU 80386

1.4.3. Các chế độ vận hành của bộ xử lý 80386

CPU-80386 có thể vận hành theo một trong ba chế độ khác nhau: chế độ thực (real mode), chế độ bảo vệ (protected mode) và chế độ 8086 ảo (virtual 8086 mode). Chế độ vận hành của CPU phải được thiết lập trước bằng phần cứng.

- Chế độ thực: chế độ thực của bộ xử lý 80386 hoàn toàn tương thích với chế độ vận hành của bộ xử lý 8086. Trong chế độ này, không gian địa chỉ của 80386 bị giới hạn ở mức 220 = 1MB giống như không gian địa chỉ của 8086 mặc dù bus địa chỉ của 80386 có 32 đường dây.
- Chế độ bảo vệ: (Còn gọi là chế độ đa nhiệm) chế độ bảo vệ đã được đầu tiên đưa vào bộ xử lý 80286. Chế độ này cho phép bộ xử lý 80386 dùng hết không gian địa chỉ của nó là 232 = 4096 MB và cho phép nó vận hành dưới một hệ điều hành đa nhiệm. Trong hệ điều hành đa nhiệm, nhiều tiến trình có thể chạy đồng thời và được bảo vệ chống lại các thâm nhập trái phép vào vùng ô nhớ bị cấm.

Trong chế độ bảo vệ, các thanh ghi đoạn không được xem như địa chỉ bắt đầu của đoạn mà là thanh ghi chọn (selector) gán các ưu tiên khác nhau cho các tiến trình. Phần ưu tiên khác nhau cho các tiến trình. Phần cốt lõi của hệ điều hành có ưu tiên cao nhất và người sử dụng có ưu tiên thấp nhất.

- Chế độ 8086 ảo: Chế độ này cho phép thiết lập một kiểu vận hành đa nhiệm trong đó các chương trình dùng trong chế độ thực, có thể chạy song song với các tiến trình khác.

1.4.4. Bộ xử lý Intel 80486:

CPU-80486DX được phát hành năm 1989. Đó là bộ xử lý 32bit chứa 1.2 triệu transistor. Khả năng quản lý bộ nhớ tối đa giống như 80386 nhưng tốc độ thi hành lệnh đạt được 26.9 MIPS (Mega Instructions Per Second - triệu lệnh mỗi giây) tại xung nhịp 33 MHz

Nếu bộ xử lý 80386 là bộ xử lý CISC thuần túy với bộ đồng xử lý toán học 80387 nằm bên ngoài bộ xử lý 80386, thì bộ xử lý 80486 là một bộ xử lý hỗn tạp CISC và RISC với bộ đồng xử lý toán học và với 8K cache nằm bên trong bộ xử lý 80486.

Trong bộ xử lý 80486, một số lệnh thường dùng, ví dụ như lệnh MOV, dùng mạch điện (kỹ thuật RISC) để thực hiện lệnh thay vì dùng vi chương trình như trong các CPU CISC thuần túy. Như thế thì các lệnh thường dùng này được thi hành với tốc độ nhanh hơn. Kỹ thuật ống dẫn cũng được đưa vào trong bộ xử lý 80486.

Với các kỹ thuật RISC được đưa vào, bộ xử lý 80486 nhanh hơn bộ xử lý 80386 đến 3 lần (nếu tốc độ xung nhịp là như nhau).

Bộ xử lý 80486 hoàn toàn tương thích với 2 bộ xử lý 80386 và 80387 cộng lại và như thế nó có các chế độ vận hành giống như 80386.

Bộ xử lý 80486 tỏ ra rất mạnh đối với các chương trình cần tính toán nhiều và các chương trình đồ họa, vì bộ đồng xử lý toán học nằm ngay trong bộ xử lý 80486. Hàng chờ lệnh của bộ xử lý 80486 là 32 byte.





Hình 1.7: CPU-80386 Hình 1.8: CPU-80486

1.4.5. Bộ xử lý Intel PENTIUM:

Được đưa ra thị trường vào giữa năm 1993, các bộ xử lý Pentium được chế tạo với hơn 3 triệu transistor, thụ hưởng các tính năng kỹ thuật của các bộ xử lý 80486 DX/2 và được có thêm nhiều tính năng mới.

Theo hãng sản xuất Intel, nếu hoạt động ở với xung nhịp 66 MHz thì tốc độ thi hành lệnh của Pentium là 112 MIPS thay vì 54 MIPS cho bộ xử lý 80486 DX/2 cùng xung nhịp.

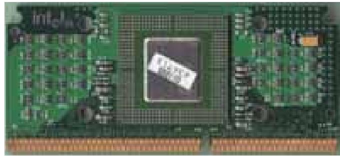
Các tính năng nổi bật của bộ xử lý Pentium là :

- Hoàn toàn tương thích với các bộ xử lý được chế tạo trước đó (8086, 80286, 80386, 80486).
- Dùng kỹ thuật ống dẫn tốt hơn với 2 ống dẫn số nguyên độc lập và một ống dẫn số lẻ. Kỹ thuật siêu vô hướng và tiên đoán lệnh nhảy cũng được đưa vào.
- Dùng cache lệnh và cache dữ liệu riêng biệt.
- Bus số liệu là 64 bit với cách vận chuyển theo từng gói.
- Dùng cách quản lý hệ thống cho phép tiết kiệm năng lượng tiêu hao trong bộ xử lý Pentium.
- Có tối ưu hóa các chuỗi mã lệnh.

Petium MMX: Các lệnh xử lý multimedia được đưa vào tập lệnh của CPU nên việc thi hành chương trình multimedia được cải thiện rất nhiều.

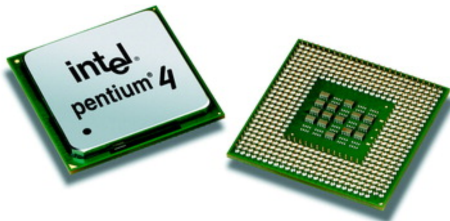


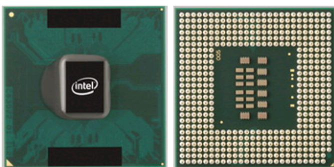






Pentium Celeron







Hình 1.9: Hình ảnh các loại bộ xử lý Intel

BÀI TẬP CHƯƠNG 1

1. Chọn các thanh ghi đa dụng để lưu trữ các dữ liệu sau đây, sao cho mỗi thanh ghi lưu trữ 1 giá trị và không trùng nhau (Giải thích việc chọn thanh ghi): 15h, 0AFh, 01234h, 230, 257, 'H', 8086.

Ghi chú: Số có tận cùng bằng h (hay H) là số thập lục phân (Hexa);

'H' : Ký tự H

2. Các thanh ghi đang lưu trữ giá trị như sau:

AH = 11hAL = 22hCL = 15

CH = 10BX = 0A1D4hDX = 8086

Hãy cho biết giá trị thập lục phân của những thanh ghi sau và giải thích:

AX, CX, BH, BL, DH, DL

3. Mô tả các cách có thể sử dụng được để lưu trữ giá trị vào thanh ghi sau:

a. 1234h vào thanh ghi SIb. 5678h vào thanh ghi AX

c. 100 vào thanh ghi DId. 100 vào thanh ghi DX

4. Sử dụng mô hình bộ nhớ gồm 17 ô nhớ như hình A1 để ghi các dữ liệu sau đây vào bộ nhớ sao cho các giá trị không chồng lên nhau (sinh viên tự chọn địa chỉ ô nhớ để lưu trữ): 15h, 0AFh, 01234h, 230, 257, 'H', 8086, 3A4B5h, 0F1D2E3h

Ô nhớ	Địa chỉ
	00010h
…..	…..
	00002h
	00001h
	00000h

Hình A1: Mô hình bộ nhớ

5. Với mô hình bộ nhớ kết quả của câu 4, hãy cho biết giá trị dữ liệu 8/16/24/32 bit tại mỗi địa chỉ sau đây ở dạng thập lục phân:

a. 00004h b. 00008hc. 0000Bh d. 0000Dh

6. Đổi sang địa chỉ vật lý tương ứng với mỗi địa chỉ logic sau:

a. 0000:0001hb. 0100:1234h

c. ABCD:3AFFhd. AF70:00CFh

7. Viết ra 4 địa chỉ logic khác nhau đối với mỗi địa chỉ vật lý sau:

a. 40000hb. 0D32FCh

8. Tìm tất cả các địa chỉ logic khác nhau có thể có của mỗi ô nhớ có địa chỉ vật lý sau đây:

a. 00000h b. 0000Fhc. 00010h d. 0001Fh

Có nhận xét gì về các kết quả trên?

9. Cho một chương trình bao gồm 100 byte lệnh (Code), 200 byte dữ liệu (Data) và 16 KB ngăn xếp (Stack). Vẽ hình mô tả tổ chức vùng nhớ của chương trình trên trong bộ nhớ có mô hình như hình A2 theo hai cách sau (sinh viên tự gán địa chỉ vật lý và logic thích hợp với đầu và cuối cho từng vùng):

a. Dùng chung 1 đoạn duy nhất cho cả 3 vùng Code, Data và Stack

b. Dùng 3 đoạn riêng biệt không chồng nhau cho Code, Data, Stack

Địa chỉ logic	Ô nhớ	Địa chỉ vật lý
….:….		….
…..:…	…..	…..
….:…		….
….:…		….
….:…		….

Hình A2: Mô hình tổ chức bộ nhớ

10. Bằng mô hình bộ nhớ kết quả của câu 9a, hãy thiết lập giá trị các thanh ghi sao cho CPU-8086 truy xuất được những ô nhớ trong mỗi trường hợp sau:

a. Byte lệnh đầu tiênb. Byte lệnh thứ 20

c. Byte dữ liệu đầu tiênd. Byte dữ liệu thứ 100

Lập trình hàm

TỔNG QUAN

Mục tiêu

Sau khi học xong chương này, sinh viên cần phải nắm:

- Khái niệm về lập trình hàm.
- Kỹ thuật lập trình đệ qui.
- Các cấu trúc cơ bản của ngôn ngữ LISP

Nội dung cốt lõi

- Lập trình hàm.
- Căn bản về ngôn ngữ lập trình LISP.

Kiến thức cơ bản cần thiết

Kiến thức và kĩ năng lập trình căn bản.

NGÔN NGỮ LẬP TRÌNH HÀM

Giới thiệu

Hầu hết các ngôn ngữ lập trình từ trước đến nay được xây dựng dựa trên nguyên lý kiến trúc máy tính Von Neumann. Lớp chủ yếu trong các ngôn ngữ đó là các ngôn ngữ ra lệnh. Đơn vị làm việc trong một chương trình là câu lệnh. Kết quả của từng câu lệnh được tổ hợp lại thành kết quả của cả chương trình. Các ngôn ngữ này bao gồm: FORTRAN, COBOL, Pasacl, Ada... Mặc dù ngôn ngữ ra lệnh đã được hầu hết người lập trình

chấp nhận nhưng sự liên hệ chặt chẽ với kiến trúc máy tính là một hạn chế đến việc phát triển phần mềm.

Ngôn ngữ lập trình hàm được thiết kế dựa trên các hàm toán học là một trong những ngôn ngữ không ra lệnh quan trọng nhất. Trong đó LISP là một ngôn ngữ tiêu biểu.

Hàm toán học

Hàm là một sự tương ứng giữa các phần tử của một tập hợp (miền xác định) với các phần tử của một tập hợp khác (miền giá trị). Định nghĩa hàm xác định miền xác định, miền giá trị và quy tắc tương ứng giữa các phần tử của miền xác định với các phần tử của miền giá trị. Thông thường sự tương ứng được mô tả bởi một biểu thức. Hàm toán học có hai đặc trưng cơ bản là:

- Thứ tự đánh giá biểu thức được điều khiển bởi sự đệ quy và biểu thức điều kiện chứ không phải bằng cách lặp lại và liên tiếp như trong các ngôn ngữ ra lệnh.
- Hàm toán học không có hiệu ứng lề cho nên với cùng một tập đối số, hàm toán học luôn cho cùng một kết quả.

Định nghĩa hàm thường được viết bởi tên hàm, danh sách các tham số nằm trong cặp dấu ngoặc và sau đó là biểu thức, ví dụ: lap_phuong(x) x*x*x trong đó x là một số thực. Miền xác định, miền giá trị là các tập số thực.

Lúc áp dụng, một phần tử cụ thể của miền xác định gọi là đối sẽ thay thế cho tham số trong định nghĩa hàm. Kết quả hàm thu được bằng cách đánh giá biểu thức hàm. Ví dụ lap_phuong(2.0) cho giá trị là 8.0. Trong định nghĩa hàm, x đại diện cho mọi phần tử của miền xác định. Trong lúc áp dụng, nó được cho một giá trị (chẳng hạn 2.0), giá trị của nó không thay đổi sau đó. Điều này trái ngược với biến trong lập trình có thể nhận các giá trị khác nhau trong quá trình thực hiện chương trình.

Trong định nghĩa hàm, ta bắt cặp tên hàm với biểu thức x*x*x. Đôi khi người ta sử dụng hàm không tên, trong trường hợp đó người ta sử dụng biểu thức lambda. Giá trị của biểu thức lambda chính là hàm của nó. Ví dụ (x)x*x*x. Tham số trong biểu thức lambda được gọi là biến kết ghép. Khi biểu thức lambda được đánh giá đối với một tham số đã cho, người ta nói rằng biểu thức được áp dụng cho tham số đó.

Dạng hàm

Dạng hàm là sự tổ hợp của các hàm. Dạng hàm phổ biến nhất là hàm hợp. Nếu f được định nghĩa là hàm hợp của g và h, được viết là f g.h thì việc áp dụng f được định nghĩa là sự áp dụng h sau đó áp dụng g lên kết quả.

Xây dựng (construction) là một dạng hàm mà các tham số của chúng là những hàm. Người ta ký hiệu một xây dựng bằng cách để các hàm tham số vào trong cặp dấu ngoặc vuông. Khi áp dụng vào một đối số thì các hàm tham số sẽ được áp dụng vào đối đó và tập hợp các kết quả vào trong một danh sách. Ví dụ: G(x) x*x, H(x) 2*x và I(x) x/2 thì [G,H,I] (4) có kết quả là (16,8,2).

Áp dụng cho tất cả là một dạng hàm mà nó lấy một hàm đơn như là một tham số. Áp dụng cho tất cả được ký hiệu là . Nếu áp dụng vào một danh sách các đối thì áp dụng cho tất cả sẽ áp dụng hàm tham số cho mỗi một giá trị và tập hợp các kết quả vào trong một danh sách. Ví dụ

Cho h(x) x*x thì (h, (2,3,4)) có kết quả là (4,9,16)

Bản chất của ngôn ngữ lập trình hàm

Mục đich của việc thiết kế ngôn ngữ lập trình hàm là mô phỏng các hàm toán học một cách nhiều nhất có thể được. Trong ngôn ngữ ra lệnh, một biểu thức được đánh giá và kết quả của nó được lưu trữ trong ô nhớ

được biểu diễn bởi một biến trong chương trình. Ngược lại, trong ngôn ngữ lập trình hàm không sử dụng biến và do đó không cần lệnh gán. Điều này giải phóng người lập trình khỏi mối quan tâm về ô nhớ của máy tính trong khi thực hiên chương trình. Không có biến cho nên không có cấu trúc lặp (vì cấu trúc lặp được điều khiển bởi biến). Các lệnh lặp lại sẽ được xử lý bằng giải pháp đệ quy. Chương trình là các định nghĩa hàm và các áp dụng hàm. Sự thực hiện là việc đánh giá các áp dụng hàm. Sự thực hiện một hàm luôn cho cùng một kết quả khi ta cho nó cùng một đối số. Điều này gọi là trong suốt tham khảo (referential transparancy). Nó cho thấy rằng ngữ nghĩa của ngôn ngữ lập trình hàm đơn giản hơn ngữ nghĩa của ngôn ngữ lập trình ra lệnh và ngôn ngữ hàm bao gồm cả những nét đặc biệt của ngôn ngữ ra lệnh.

Ngôn ngữ hàm cung cấp một tập hợp các hàm nguyên thủy, một tập các dạng hàm để xây dựng các hàm phức tạp từ các hàm đã có. Ngôn ngữ cũng cung cấp một phép toán áp dụng hàm và các cấu trúc lưu trữ dữ liệu. Một ngôn ngữ hàm được thiết kế tốt là một ngôn ngữ có tập hợp nhỏ các hàm nguyên thủy. Phần sau chúng ta làm quen với một ngôn ngữ lập trình hàm khá nổi tiếng là ngôn ngữ LISP.

NGÔN NGỮ LISP

Giới thiệu:

Được J. MAC CARTHY viết năm 1958, LISP là một trong những ngôn ngữ lập trình sớm nhất. Đầu năm những năm 80, LISP được phát triển mạnh nhờ những áp dụng trong lĩnh vực trí tuệ nhân tạo. LISP có các ưu điểm chính như sau:

- Cú pháp đơn giản. Trong LISP chỉ có một cấu trúc duy nhất là cấu trúc danh sách (LISP là ngôn ngữ xử lý danh sách: LISP = LISt Processing language), không có lệnh, không có từ khóa, tất cả các hàm đều được viết dưới dạng danh sách.
- Là một ngôn ngữ mạnh nhờ tính tương đương giữa dữ liệu và chương trình: dữ liệu và chương trình đều là danh sách, chúng có thể

thao tác nhờ chung một công cụ.
- Mềm dẻo và dễ phát triển.

Các khái niệm cơ bản

Nguyên tử (atom)

Nguyên tử là một đối tượng cơ bản của LISP, nguyên tử có thể là số hoặc ký hiệu.

- Số. Dữ liệu số trong LISP cũng giống như trong một số ngôn ngữ lập trình khác như Pascal, C…

Ví dụ về các hằng số: 5, -17, 5.35, 3/4, 118.2E+5,...

- Ký hiệu (symbol) là một chuỗi các ký tự (trừ các ký tự đặc biệt, dấu ngoặc và khoảng trống). Các hằng ký hiệu được viết mở đầu bằng dấu nháy đơn ‘.

Ví dụ về các hằng ký hiệu: ‘a, ‘anh, ‘anh_ba,...

Một số ký hiệu được định nghĩa trước như: T (về mặt logic, được hiểu là TRUE), NIL (về mặt logic, được hiểu là FALSE).

Hằng ký hiệu số được xem như là một số, chẳng hạn ‘5 = 5.

Danh sách

Danh sách là một dãy có phân biệt thứ tự của các phần tử cách nhau ít nhất một khoảng trắng và đặt nằm trong cặp dấu ngoặc đơn ().

Phần tử của danh sách có thể là một nguyên tử hoặc là một danh sách.

Hằng danh sách được mở đầu bằng dấu nháy đơn ‘.

Ví dụ về các hằng danh sách:

- ‘()Danh sách rỗng, tương đương ký hiệu NIL.
- ‘(a 5 c)Danh sách gồm 3 phần tử.
- ‘(3 (b c) d (e (f g)))Danh sách gồm 4 phần tử, trong đó phần tử thứ 2 và phần tử thứ 4 lại là các danh sách.

Biểu thức

Biểu thức là một nguyên tử hoặc một danh sách. Biểu thức luôn có một giá trị mà việc định trị nó theo nguyên tắc sau:

- Nếu biểu thức là một số, thì giá trị của biểu thức là giá trị của số đó.

Ví dụ:

> 25

= 25

- Nếu biểu thức là một ký hiệu thì giá trị của biểu thức có thể là

- Được xác định trước bởi LISP (chẳng hạn t có giá trị là T (TRUE) và nil có giá trị là NIL một danh sách rỗng) hoặc
- Một giá trị dữ liệu của người sử dụng hoặc trong chương trình được gán cho một biến. Biến không cần phải khai báo.

Ví du:

> (setq a 3) ; Gán số 3 cho biến có tên a

= 3

> a ; hỏi giá trị của ký hiệu “a”

= 3

- Nếu biểu thức là một danh sách có dạng (E0 E1 ... En) thì giá trị của biểu thức được xác định theo cách sau đây:

- Phần tử đầu tiên E0 phải là một hàm đã được LISP nhận biết.
- Các phần tử E1, E2, ..., En được định trị tuần tự từ trái sang phải. Giả sử ta có các giá trị tương ứng là V1, V2, ..., Vn
- Hàm E0 được áp dụng cho các đối V1, V2, ..., Vn. Giá trị của hàm E0 chính là giá trị của biểu thức.

Ví dụ

> (+ 5 3 6)

= 14

> (+ 4 (+ 3 5))

= 12

- Chú ý: Nếu biểu thức dùng hàm QUOTE hoặc dấu nháy đơn sẽ không được đánh giá

Ví dụ:

> ‘(+ 1 2)

= (+ 1 2)

Các hàm

Một chương trình của LISP là một hàm hoặc một hàm hợp. Các hàm có thể do LISP định nghĩa trước hoặc do lập trình viên tự định nghĩa.

Một số hàm định nghĩa trước

- Các hàm số học: +, -, *, /, 1+, 1-, MOD, SQRT tác động lên các biểu thức số và cho kết quả là một số.

Ví dụ:

> (+ 5 6 2)

= 13

> (- 8 3)

= 5

> (- 8 3 1)

= 4

>(1+ 5) ; Tương đương (+ 5 1)

= 6

> (1- 5) ; Tương đương (- 5 1)

= 4

>(MOD 14 3)

= 2

>(sqrt 9) ; Lấy căn bậc hai của 9

= 3

- Các hàm so sánh các số <, >, <=, >=, = và /=, cho kết quả là T hoặc NIL

Ví dụ:

>(< 4 5)

= T

>(> 4 (* 2 3))

= NIL

- (EQ s1 s2) so sánh xem hai ký hiệu s1 và s2 có giống nhau hay không?

Ví dụ:

>(eq ‘tuong ‘tuong)

= T

>(eq ‘tuong ‘duong)

= NIL

>(eq ‘5 5)

= T

- (EQUAL o1 o2) so sánh xem đối tượng bất kỳ o1 và o2 có giống nhau hay không?

Ví dụ:

>(equal ‘(a b c) ‘(a b c))

= T

>(equal ‘(a b c) ‘(b a c))

= NIL

>(equal ‘a ‘a)

= T

- Các hàm thao tác trên danh sách: CAR, CDR, CONS và LIST
- (CAR L) nhận vào danh sách L, trả về phần tử đầu tiên của L.

Ví du:

> (CAR '(1 2 3))

= 1

> (CAR 3)

Error: bad argument type - 3

>(CAR nil)

= NIL

> (CAR '((a b) 1 2 3))

= (A B)

- (CDR L) nhận vào danh sách L, trả về một danh sách bằng phần còn lại của danh sách L sau khi bỏ đi phần tử đầu tiên.

Ví dụ:

>(cdr '(1 2 3))

= (2 3)

>(cdr 3)

Error: bad argument type - 3

>(cdr nil)

= NIL

>(cdr '(1))

= NIL

>(CAR (CDR ‘(a b c)))

= B

- Viết gộp các hàm: Ta có thể dùng hàm C..A/D..R để kết hợp nhiều CAR và CDR (có thể thay thế việc lồng nhau tới 4 cấp)

Ví du:

(CADR ‘(a b c))

= B

- (CONS x L) nhận vào phần tử x và danh sách L, trả về một danh sách, có được bằng cách thêm phần tử x vào đầu danh sách L

Ví du:

>(CONS 3 '(1 2 3))

= (3 1 2 3)

>(CONS 3 nil)

= (3)

>(CONS '(a b) '(1 2 3))

= ((A B) 1 2 3)

- (LIST E1 E2 ... En) nhận vào n biểu thức E1, E2, ..., En, trả về danh sách bao gồm n phần tử V1, V2, ..., Vn, trong đó Vi là giá trị của biểu thức Ei (i=1..n) .

Ví du:

>(list 1 2)

= (1 2)

>(list 'a 'b)

= (A B)

>(list 'a 'b (+ 2 3 5))

= (A B 10)

- Các vị từ kiểm tra

- (ATOM a) xét xem a có phải là một nguyên tử.

- (NUMBERP n) xét xem n có phải là một số.
- (LISTP L) xét xem L có phải là một danh sách.
- (SYMBOLP S) xét xem S có phải là một ký hiệu.
- (NULL L) nhận vào 1 danh sách L. Nếu L rỗng thì trả về kết quả là T, ngược lại thì trả về kết quả là NIL.

Ví du:

>(atom 'a)

= T

>(numberp 4)

= T

>(symbolp 'a)

= T

>(listp '(1 2))

= T

>(symbolp NIL)

= T

>(listp NIL)

= T

>(null NIL)

= T

>(null ‘(a b))

= NIL

>(null 10)

= NIL

- Các hàm logic AND, OR và NOT
- (AND E1 E2... En) nhận vào n biểu thức E1, E2,... En. Hàm AND định trị các biểu thức E1 E2... En từ trái sang phải. Nếu gặp một biểu thức là NIL thì dừng và trả về kết quả là NIL. Nếu tất cả các biểu thức đều khác NIL thì trả về giá trị của biểu thức En.

Ví dụ:

>(AND (> 3 2) (= 3 2) (+ 3 2))

= NIL

>(AND (> 3 2) (- 3 2) (+ 3 2))

= 5

- (OR E1 E2 ... En) nhận vào n biểu thức E1, E2,... En. Hàm OR định giá các biểu thức E1 E2... En từ trái sang phải. Nếu gặp một biểu thức khác NIL thì dừng và trả về kết quả là giá trị của biểu thức đó. Nếu tất cả các biểu thức đều là NIL thì trả về kết quả là NIL.

Ví du:

>(OR (= 3 2) (+ 2 1) (list 1 2))

= 3

>(OR (= 2 1) (Cdr ‘(a)) (listp 3))

= NIL

- (NOT E) nhận vào biểu thức E. Nếu E khác NIL thì trả về kết quả là NIL, ngược lại thì trả về kết quả là T.

- Các hàm điều khiển

- (IF E1 E2 E3) nhận vào 3 biểu thức E1, E2 và E3. Nếu E1 khác NIL thì hàm trả về giá trị của E2 ngược lại trả về giá trị của E3
- (IF E1 E2) tương đương (IF E1 E2 NIL)
- Nếu E2 khác NIL thì (IF E1 E2 E3) tương đương (OR (AND E1 E2) E3)

- (COND(ĐK1E1)

(ĐK2E2)

....

(ĐKnEn)

[(TEn+1)]

)

Nếu ĐK1 khác NIL thì trả về kết quả là giá trị của E1, ngược lại sẽ xét ĐK2. Nếu ĐK2 khác NIL thì trả về kết quả là giá trị của E2, ngược lại sẽ xét ĐK3...

......

Nếu ĐKn khác NIL thì trả về kết quả là giá trị của En, ngược lại sẽ trả về NIL hoặc trả về kết quả là giá trị của En+1 (trong trường hợp ta sử dụng (T En+1))

- (PROGN E1 E2 ... En) nhận vào n biểu thức E1, E2,... En. Hàm định trị các biểu thức E1, E2,... En từ trái sang phải và trả về kết quả là giá trị của biểu thức En.
- (PROG1 E1 E2 ... En) nhận vào n biểu thức E1, E2,... En. Hàm định trị các biểu thức E1, E2,... En từ trái sang phải và trả về kết quả là giá trị của biểu thức E1.

Hàm do người lập trình định nghĩa

Cú pháp định nghĩa hàm là:

(defun <tên hàm> <danh sách các tham số hình thức>

<biểu thức>

)

Ví dụ 1: Định nghĩa hàm lấy bình phương của số a

(defun binh_phuong (a)

(* a a)

)

Sau khi nạp hàm này cho LISP, ta có thể sử dụng như các hàm đã được định nghĩa trước.

>(binh_phuong 5)

= 25

>(binh_phuong (+ 5 2))

= 49

Ví dụ 2: Định nghĩa hàm DIV chia số a cho số b, lấy phần nguyên.

Trước hết ta có: a DIV b = (a – a MOD b)/b

(defun DIV (a b)

(/ (- a (MOD a b)) b)

)

Đệ quy

Một hàm đệ quy là một hàm có lời gọi chính nó trong biểu thức định nghĩa hàm. Mô tả một đệ quy bao gồm:

- Có ít nhất một trường hợp "dừng" để kết thúc việc gọi đệ quy.
- Lời gọi đệ quy phải bao hàm yếu tố dẫn đến các trường hợp "dừng".

Ví dụ 1: Viết hàm tính n giai thừa

Công thức đệ quy tính n giai thừa là $\mathrm{n!} = \begin{cases} 1 \text{ neu n} = 0 \\ \mathrm{n}*(n-1)! \end{cases}$

Hàm (giai_thua N) viết bằng ngôn ngữ LISP:

(defun giai_thua (n)

(if (= n 0) 1 ; trường hợp “dừng”

(* n (giai_thua (1- n))); n-1 là yếu tố dẫn đến trường hợp dừng

) ; If

)

Ví dụ 2: Viết hàm DIV chia a cho b lấy phần nguyên, viết bằng đệ quy.

Công thức đệ quy: $\text{a DIV b} = \begin{cases} 0 \text{ neu a} < b \\ 1 + (a-b)\text{DIV}b \end{cases}$

Hàm (DIV a b) viết bằng LISP:

(defun DIV (a b)

(if (< a b) 0 ; Trường hợp “dừng”

(1+ (DIV (- a b) b)); a-b là yếu tố dẫn đến trường hợp dừng

) ; If

)

Ví dụ 3: Viết hàm (phan_tu i L), nhận vào số nguyên dương i và danh sách L. Hàm trả về phần tử thứ i trong danh sách L hoặc thông báo “không tồn tại”.

Công thức đệ quy:

$$\text{Phan tu thu i trong DS L} = \begin{matrix} Khong\,ton\,tai \text{ neu DS L rong} \\ \text{Phan tu dau tien cua L neu i} = 1 \\ \text{Phan tu thu}(i-1)\text{trong DS } duoi \text{ cua L} \end{matrix}$$

Hàm (phan_tu i L) viết bằng LISP:

```
(defun phan_tu(i L)
(cond
((Null L) “Khong ton tai”)
((= i 1) (car L)); trường hợp dừng thứ hai
(T (phan_tu (1- i) (cdr L)))
) ; cond
)
```

Trong chương trình trên, (null L) là trường hợp “dừng” thứ nhất; (= i 1) là trường hợp “dừng” thứ hai; (cdr L) là yếu tố dẫn đến trường hợp “dừng” thứ nhất và (1- i) yếu tố dẫn đến trường hợp “dừng” thứ hai.

Các hàm nhập xuất

- (LOAD <Tên tập tin>)

Nạp một tập tin vào cho LISP và trả về T nếu việc nạp thành công, ngược lại trả về NIL. Tên tập tin là một chuỗi kí tự có thể bao gồm cả đường dẫn đến nơi lưu trữ tập tin đó. Tên tập tin theo quy tắc của DOS, nghĩa là chỉ có tối đa 8 ký tự trong phần tên và 3 ký tự phần mở rộng và không chứa các ký tự đặc biệt.

Ta có thể sử dụng LOAD để nạp một tập tin chương trình của LISP trước khi gọi thực hiện các hàm đã được định nghĩa trong tập tin đó.

Ví dụ:

>(Load “D:\btlisp\bai1.lsp”)

- (READ)

Đọc dữ liệu từ bàn phím cho đến khi gõ phím Enter, trả về kết quả là dữ liệu được nhập từ bàn phím.

- (PRINT E)

In ra màn hình giá trị của biểu thức E, xuống dòng và trả về giá trị của E.

- (PRINC E)

In ra màn hình giá trị của biểu thức E (không xuống dòng) và trả về giá trị của E.

- (TERPRI)

Đưa con trỏ xuống dòng và trả về NIL.

Biến toàn cục và biến cục bộ

Biến toàn cục

Biến toàn cục (global variables) là biến mà phạm vi của nó là tất cả các hàm. Biến toàn cục sẽ tự động giải phóng khi chương trình dịch LISP kết thúc.

- Hàm (SETQ <tên biến> <biểu thức>)

Gán trị của <biểu thức> cho <tên biến> và trả về kết quả là giá trị của <biểu thức>.

Ví dụ:

>(setq x (* 2 3))

= 6

> x ; biến x vẫn còn tồn tại và có giá trị là 6

= 6

Biến cục bộ

Biến cục bộ (local variables) là biến mà phạm vi của nó chỉ nằm trong hàm mà nó được tạo ra. Biến cục bộ sẽ tự động giải phóng hàm tạo ra nó kết thúc.

- (LET ((var1 E1) (var2 E2) ... (vark Ek)) Ek+1 ... En)

Ta thấy hàm này có 2 phần: phần gán trị cho các biến và phần định trị các biểu thức.

Gán trị của biểu thức Ei cho biến cục bộ vari tương ứng và thực hiện (PROGN Ek+1 ... En).

Ví dụ:

>(Let ((a 3) (b 5)) (* a b) (+ a b))

= 8

> a ; biến a lúc này đã được giải phóng nên LISP sẽ thông báo lỗi

error: unbound variable - A

Biến cục bộ che biến toàn cục

Trong lập trình hàm, người ta rất hạn chế sử dụng biến, nếu thật sự cần thiết thì nên sử dụng biến cục bộ. Tuy nhiên việc khai báo biến cục bộ trong hàm LET gây khó khăn cho việc viết chương trình hơn là sử dụng biến toàn cục. Để khắc phục tình trạng này, ta sẽ kết hợp cả hai hàm

LET và SETQ để sử dụng biến cục bộ che biến toàn cục. Cách làm như sau:

- Trong phần gán trị cho biến của LET ta tạo ra một biến và gán cho nó một giá trị bất kỳ, chẳng hạn số 0.
- Trong phần định trị các biểu thức, ta có thể sử dụng SETQ để gán trị cho biến đã tạo ra ở trên, biến này sẽ là một biến cục bộ chứ không còn là toàn cục nữa.
- Cụ thể chúng ta có thể viết:

(LET ((var E1)…..)

…….

(SETQ var E2)

……

)

Với cách làm này thì biến var trong hàm SETQ sẽ trở thành biến cục bộ.

Ví dụ: Giả sử ta đã định nghĩa được hàm (ptb2 a b c), giải phương trình bậc hai ax2+bx+c = 0. Bây giờ ta viết hàm (giai_ptb2) cho phép nhập các hệ số a, b, c từ bàn phím và gọi hàm (ptb2 a b c) để thực hiện việc giải phương trình. Có hai phương pháp để viết hàm này.

Phương pháp 1: dùng các biến toàn cục a, b, c

(defun giai_ptb2 ()

(progn

(print "Chương trình giải phương trình bậc hai")

(princ "Nhập hệ số a: ") (setq a (read))

(princ "Nhập hệ số b: ") (setq b (read))

(princ “Nhập hệ số c: “) (setq c (read))

(ptb2 a b c)

)

)

Sau khi thực hiện chương trình này, thì các biến toàn cục a, b và c vẫn còn.

Phương pháp 2: dùng các biến cục bộ d, e, f

(defun giai_ptb2 ()

(let ((d 0) (e 0) (f 0))

(print “Chương trình giải phương trình bậc hai“)

(princ “Nhập hệ số a: “) (setq d (read))

(princ “Nhập hệ số b: “) (setq e (read))

(princ “Nhập hệ số c: “) (setq f (read))

(ptb2 d e f)

)

)

Sau khi thực hiện chương trình này, thì các biến cục bộ d, e và f được giải phóng.

Hướng dẫn sử dụng LISP

Sử dụng XLISP

XLISP là một trình thông dịch, chạy dưới hệ điều hành Windows. Chỉ cần chép tập tin thực thi XLISP.EXE có dung lượng 288Kb vào máy tính của bạn là có thể thực hiện được.

Để thực hiện các hàm, chỉ cần gõ trực tiếp hàm đó vào sau dấu chờ lệnh (>) của XLISP. Trong trường hợp không có dấu chờ lệnh, hãy dùng menu Run/Top level hoặc Ctrl-C để làm xuất hiện dấu chờ lệnh.

Việc định nghĩa một hàm cũng có thể gõ trực tiếp vào sau dấu chờ lệnh. Tuy nhiên cách làm này sẽ khó sửa chữa hàm đó và do vậy ta thường định nghĩa các hàm trong một tập tin chương trình, sau đó nạp vào cho XLISP để sử dụng.

Ta có thể lưu trữ lại tình trạng làm việc hiện hành vào trong tập tin .WKS bằng cách dùng menu File/Save workspace và sau đó có thể khôi phục lại bằng cách dùng menu File/Restore workspace.

Soạn thảo tập tin chương trình

Do XLISP không có công cụ để soạn thảo chương trình nên ta có thể sử dụng Notepad để soạn thảo tập tin chương trình.

Trong một tập tin chương trình ta có thể định nghĩa nhiều hàm.

Lưu tập tin chương trình có tên theo quy định của DOS (8.3) với phần mở rộng .LSP và để trong cặp dấu nháy kép.

Nạp hàm tự định nghĩa cho XLISP

Có hai phương pháp để nạp các hàm tự định nghĩa cho XLISP:

- Phương pháp 1: Copy và dán khối

- Trong Notepad, đánh dấu khối một hàm tự định nghĩa và copy khối (Edit/Copy hoặc Ctrl-C).
- Trong XLISP, dán khối tại dấu chờ lệnh (Edit/Paste hoặc Ctrl-Ins).
- Với phương pháp này thì khi viết các hàm, không nên viết một dòng lệnh quá dài.
- Nếu khối hàm dán vào không có lỗi thì tên hàm sẽ xuất hiện và ta có thể sử dụng được hàm đó.
- Phương pháp này rất phù hợp với việc kiểm thử từng hàm.

- Phương pháp 2: Mở tập tin chương trình

- Trong XLISP, sử dụng menu File-Open/Load để mở tập tin chương trình chứa các hàm đã được viết và lưu trữ bởi Notepad. Chúng ta cũng có thể sử dụng hàm (LOAD <tên tập tin>) để mở tập tin chương trình.
- Nếu việc mở thành công thì có thể gọi thực hiện bất kỳ hàm nào đã có trong tập tin chương trình.
- Nếu có một hàm viết sai dấu ngoặc thì việc mở tập tin sẽ thất bại và do đó ta không thể dùng bất kỳ hàm nào trong tập tin đó.
- Phương pháp này thích hợp với việc nạp nhiều hàm đã được kiểm chứng trong một tập tin chương trình để sử dụng.

Một số thông báo lỗi thường gặp

- Unbound function: Hàm không có.
- Bad function: Hàm sai.
- Too many arguments: Thừa tham số.
- Too few arguments: Thiếu tham số.
- Misplaced close paren: Thừa dấu ngoặc đóng/ Thiếu dấu ngoặc mở.
- EOF reached beore expression end: Thừa dấu ngoặc mở/ Thiếu dấu ngoặc đóng.
- Not a number: Đối số của hàm phải là một số.
- Bad argument type: Kiểu của tham số sai.

Lập trình logic

TỔNG QUAN

Mục tiêu

Sau khi học xong chương này, sinh viên cần phải nắm:

- Khái niệm về lập trình logic.
- Các nguyên tắc trong lập trình logic.
- Viết chương trình đơn giản bằng ngôn ngữ Prolog.

Nội dung cốt lõi

- Lập trình logic.
- Căn bản về ngôn ngữ lập trình Prolog.

Kiến thức cơ bản cần thiết

Kiến thức và kĩ năng lập trình căn bản

GIỚI THIỆU VỀ LẬP TRÌNH LOGIC

Trong lập trình logic, ta có thể sử dụng các vị từ để định nghĩa các khái niệm của tất cả các môn khoa học khác.

Ví dụ định nghĩa một số nguyên tố:

Số nguyên tố N là một số nguyên lớn hơn 1, chỉ chia hết cho 1 và chính nó.

Để xét xem số N có phải là số nguyên tố hay không, người ta thường sử dụng dấu hiệu nhận biết: Số nguyên tố là một số nguyên dương, không chia hết cho mọi số nguyên tố nhỏ hơn nó và 2 là số nguyên tố nhỏ nhất.

Dấu hiệu này có thể mô tả bằng các vị từ như sau:

- 2 là một số nguyên tố.
- N là một số nguyên tố nếu N>0, M là số nguyên tố nào đó, M<N và N không chia hết cho M.

Khi mô tả bài toán dưới dạng logic vị từ, ta có thể yêu cầu hệ thống tìm kiếm các lời giải liên quan đến các khai báo đó. Bài toán cần giải được xem là “mục tiêu” mà hệ thống phải chứng minh trên cơ sở các tri thức đã được khai báo.

Như thế, toàn bộ các ký hiệu của ngôn ngữ lập trình suy về một công thức đặc biệt:

- Phát sinh từ một yêu cầu.
- Nhằm chứng minh một mục tiêu. Để trả lời cho câu hỏi đó hệ thống xem nó như là một “đích” và cố chứng minh “đích” đó bằng cách tạo những suy diễn trên cơ sở các tri thức đã khai báo.

Một ngôn ngữ logic có thể được dùng trong giai đoạn đặc tả yêu cầu của quy trình xây dựng một sản phẩm phần mềm. Hơn thế nữa, logic vị từ cho phép biểu diễn hầu hết các khái niệm và các định lý trong các bộ môn khoa học.

Một trong những ngôn ngữ lập trình logic có hỗ trợ rất nhiều cho lĩnh vực trí tuệ nhân tạo mà ta xét đến ở đây đó là ngôn ngữ Prolog.

NGÔN NGỮ PROLOG

Giới thiệu

Prolog là một ngôn ngữ cấp cao, có đặc điểm gần với ngôn ngữ tự nhiên, từ những người mới học đến những lập trình viên chuyên nghiệp đều có thể tiếp cận một cách nhanh chóng, viết ra một chương trình ứng dụng hữu ích.

Prolog ra đời vào năm 1973 do C.Camerauer (Đại học Marseilles, Pháp) và nhóm đồng sự phát triển. Từ đó đến nay, qua nhiều lần cải tiến, đặc biệt hãng Borland cho ra đời phần mềm TURBO PROLOG với nhiều ưu điểm, thuận tiện cho người sử dụng. Để giải quyết một số vấn đề, ta nhận thấy sử dụng ngôn ngữ Prolog cho ta chương trình gọn nhẹ hơn nhiều so với các ngôn ngữ khác.

Khác với những ngôn ngữ cấu trúc như Pascal, hay C mà ta đã làm quen, Prolog là một ngôn ngữ mô tả, với một số sự kiện và quy luật suy diễn đã mô tả, Prolog sẽ suy luận cho ta các kết quả.

Các yếu tố cơ bản của Turbo Prolog

Trong một chương trình Prolog, ta cần khai báo các yếu tố sau đây: đối tượng, quan hệ giữa các đối tượng, sự kiện và các luật.

Đối tượng

Gồm có các hằng và biến. Hằng mang giá trị cho sẵn ở đầu chương trình hoặc trong quá trình viết ta đưa vào; Các biến có giá trị thay đổi sẽ được gán giá trị khi chạy chương trình. Tên biến là một ký tự hoa hoặc một chuỗi ký tự, bắt đầu bằng một ký tự hoa.

Có một loại biến đặc biệt gọi là biến tự do, biến này không có tên và người ta dùng ký hiệu _ (dấu gạch dưới) thay cho tên biến.

Quan hệ giữa các đối tượng

Quan hệ giữa các đối tượng được dùng dưới hình thức vị từ.

Ví dụ: Thich(X,Y) là vị từ diễn tả câu “X thích Y” trong ngôn ngữ tự nhiên.

Blue(car) là vị từ diễn tả câu “Car is blue”.

Như vậy các vị từ sẽ bao gồm tên của vị từ và các đối số của nó. Các đối số được đặt trong ngoặc và phân cách nhau bởi dấu phẩy.

Sự kiện và luật

Sự kiện là một vị từ diễn tả một sự thật.

Ví dụ: “2 là một số nguyên tố” là một sự kiện vì nó diễn tả sự thật 2 là một số nguyên tố.

Luật là vị từ diễn tả quy luật suy diễn mà ta công nhận đúng. Luật được trình bày dưới dạng một mệnh đề.

Ví dụ để suy diễn số nguyên N bất kỳ là một số nguyên tố ta viết:

“N là một số nguyên tố nếu N>0, M là số nguyên tố nào đó, M<N và N không chia hết cho M”.

Cấu trúc của một chương trình Prolog

Một chương trình Prolog thường gồm có 3 hoặc 4 đoạn cơ bản: clauses, predicates, domains và goal. Phần goal có thể bỏ đi, nếu ta không thiết kế goal trong chương trình, thì khi thực hiện, hệ thống sẽ yêu cầu ta nhập goal vào.

Phần Domains

Đây là phần định nghĩa kiểu mới dựa vào các kiểu đã biết. Các kiểu được định nghĩa ở đây sẽ được sử dụng cho các đối số trong các vị từ. Nếu các vị từ sử dụng đối số có kiểu cơ bản thì có thể không cần phải định nghĩa lại các kiểu đó. Tuy nhiên để cho chương trình sáng sủa, người ta sẽ định nghĩa lại cả các kiểu cơ bản.

Cú pháp: <danh sách kiểu mới> = <kiểu đã biết> hoặc <danh sách kiểu mới> = <danh sách kiểu đã biết>

Trong đó các kiểu mới phân cách nhau bởi dấu phẩy, còn các kiểu đã biết phân cách nhau bởi dấu chấm phẩy.

Ví dụ:

Domains

ten, tac_gia, nha_xb, dia_chi = string

nam, thang, so_luong = integer

dien_tich = real

nam_xb = nxb(thang, nam)

do_vat = sach(tac_gia, ten, nha_xb, nam_xb); xe(ten, so_luong); nha(dia_chi, dien_tich)

Trong ví dụ trên, ta đã định nghĩa các kiểu mới, trong đó các kiểu mới ten, tac_gia, nha_xb, dia_chi dựa vào cùng một kiểu đã biết là string; các kiểu mới nam, thang, so_luong dựa vào cùng một kiểu đã biết là integer; kiểu mới dien_tich dựa vào kiểu đã biết là real; kiểu mới năm_xb dựa vào kiểu nxb được xây dựng từ các kiểu đã biết là thang, nam; còn kiểu do_vat lại dựa vào các kiểu sach, xe, nha mà các kiểu này lại dựa vào các kiểu đã biết.

Phần Predicates

Đây là phần bắt buộc phải có. Trong phần này chúng ta cần phải khai báo đầy đủ các vị từ sử dụng trong phần Clauses, ngoại trừ các vị từ mà Turbo Prolog đã xây dựng sẵn.

Cú pháp: <Tên vị từ> (<danh sách các kiểu>)

Các kiểu là các kiểu cơ bản hoặc là các kiểu đã được định nghĩa trong phần domains và được viết phân cách nhau bơi dấu phẩy.

Ví dụ:

Predicates

so_huu (ten, do_vat)

so_nguyen_to(integer)

Trong ví dụ trên ta khai báo hai vị từ. Trong đó vị từ so_huu (ten, do_vat) để chỉ một người có tên là ten sẽ sở hữu môt do_vat nào đó. Còn vị từ so_nguyen_to(integer) để xét xem một số integer nào đó có phải là số nguyên tố hay không.

Phần Clauses

Đây là phần bắt buộc phải có dùng để mô tả các sự kiện và các luật, sử dụng các vị từ đã khai báo trong phần predicates.

Cú pháp:

<Tên vị từ>(<danh sách các tham số>) <kí hiệu>

<Tên vị từ 1>(<danh sách các tham số 1>) <kí hiệu>

………

<Tên vị từ N>(<danh sách các tham số N>) <kí hiệu>

Trong đó: Tên vị từ phải là các tên vị từ đã được khai báo trong phần predicates. Các tham số có thể là các hằng hoặc biến có kiểu tương thích với các kiểu tương ứng đã được khai báo trong các vị từ ở trong phần predicates; các tham số được viết cách nhau bởi dấu phẩy. Các kí hiệu bao gồm:

:- (điều kiện nếu).

, (điều kiện và).

; (điều kiện hoặc).

. (kết thúc vị từ)

Ví dụ:

Clauses

so_nguyen_to(2):-!.

so_nguyen_to(N):-N>0,

so_nguyen_to(M),

M<N,

N MOD M <>0.

so_huu("Nguyen Van A", sach("Do Xuan Loi", "Cau truc DL", "Khoa hoc Ky thuat", nxb(8,1985))).

Chú ý: Nếu trong các tham số của một vị từ có biến thì biến này phải xuất hiện ít nhất 2 lần trong vị từ đó hoặc trong các vị từ dùng để suy diễn ra vị từ đó. Nếu chỉ xuất hiện một lần thì bắt buộc phải dùng biến tự do.

Ví dụ: Để diễn tả sự kiện: Tổ hợp chập 0 của N (N bất kỳ) bằng 1, ta không thể viết Tohop(N,0,1) vì biến N chỉ xuất hiện đúng một lần trong vị từ này, do đó ta phải viết Tohop(_,0,1) .

Phần Goal

Bao gồm các mục tiêu mà ta yêu cầu Turbo Prolog xác định và tìm kết quả. Đây là phần không bắt buộc phải có. Nếu ta viết sẵn trong chương trình thì đó gọi là goal nội; Nếu không, khi chạy chương trình Turbo Prolog sẽ yêu cầu ta nhập goal vào, lúc này gọi là goal ngoại.

Cú pháp phần goal giống như cú pháp phần clauses. Tức là ta đưa vào một hoặc một số các vị từ.

Nếu tất cả các tham số của vị từ là hằng thì kết quả nhận được là Yes (đúng) hoặc No (sai). Nếu trong các tham số của vị từ có biến thì kết quả trả về sẽ là các giá trị của biến.

Ngoài các phần chủ yếu nói trên, ta có thể đưa vào các phần liên quan đến khai báo hằng, các tập tin liên quan hoặc chỉ thị dịch.

Ví dụ:

Constants

Pi = 3.141592653

Một số ví dụ về chương trình prolog

Ví dụ 1: Xét xem một số N có phải là số nguyên tố hay không.

```
domains

so_nguyen = integer

predicates

so_nguyen_to(so_nguyen)

Clauses

so_nguyen_to(2):-!.

so_nguyen_to(N):-N>0,

so_nguyen_to(M),

M<N,

N MOD M <>0.

goal

so_nguyen_to(13).
```

Ví dụ 2: Giả sử ta có bảng số liệu như sau:

Tên người	giới tính	Đặc điểm	Tiêu chuẩn kết bạn
lan	nữ	đẹp, khoẻ, tốt,	khoẻ, thông minh, đẹp
hồng	nữ	đẹp, thông minh, giàu	khoẻ, thông minh, giàu
thuỷ	nữ	tốt, khoẻ, giàu	đẹp, khoẻ, thông minh
anh	nam	khoẻ, giàu, thông minh	đẹp, thông minh, tốt
bình	nam	đẹp, khoẻ, thông minh	đẹp, khoẻ
hùng	nam	giàu, thông minh, khoẻ	tốt, thông minh, khoẻ

Tiêu chuẩn kết bạn là hai người khác phái, người này hội đủ các tiêu chuẩn của người kia và ngược lại. Hãy viết chương trình để tìm ra các cặp có thể kết bạn với nhau.

domains

ten, g_tinh = symbol

predicates

gioi_tinh(ten, g_tinh)

dep(ten)

tot(ten)

giau(ten)

thong_minh(ten)

khoe(ten)

thich(ten,ten)

ket_ban(ten,ten)

clauses

gioi_tinh(lan,nu).

gioi_tinh(hong,nu).

gioi_tinh(thuy,nu).

gioi_tinh(anh,nam).

gioi_tinh(binh,nam).

gioi_tinh(hung,nam).

dep(lan).

dep(hong).

```
dep(binh).

khoe(thuy).

khoe(lan).

khoe(binh).

khoe(anh).

khoe(hung).

tot(lan).

tot(thuy).

thong_minh(hong).

thong_minh(anh).

thong_minh(hung).

thong_minh(binh).

giau(hong).

giau(thuy).

giau(hung).

thich(lan,X):-khoe(X), dep(X), thong_minh(X).

thich(hong,X):-khoe(X), thong_minh(X), giau(X).

thich(thuy,X):-khoe(X), dep(X), thong_minh(X).

thich(ann,X):-dep(X), tot(X), thong_minh(X).

thich(binh,X):-dep(X), khoe(X).

thich(hung,X):-khoe(X), tot(X), thong_minh(X).

ket_ban(X,Y):- gioi_tinh(X,M),

gioi_tinh(Y,N),

M<>N,

thich(X,Y),

thich(Y,X).
```

Các nguyên tắc của ngôn ngữ Prolog

Việc giải quyết vấn đề trong ngôn ngữ Prolog chủ yếu dựa vào hai nguyên tắc sau: Đồng nhất, quay lui.

Đồng nhất

Một quan hệ có thể đồng nhất với một quan hệ nào đó cùng tên, cùng số lượng tham số, các đại lượng con cũng đồng nhất theo từng cặp.

Một hằng có thể đồng nhất với một hằng.

Một biến có thể đồng nhất với một hằng nào đó và có thể nhận luôn giá trị hằng đó.

Chẳng hạn trong ví dụ 2 nói trên nếu ta sử dụng goal dep(lan) thì có kết quả là Yes. Nếu ta dùng goal dep(X) thì sẽ có 3 kết quả: X=lan, X=hong và X=binh.

Khi ta dùng goal dep(lan) thì dep(lan) sẽ đồng nhất với sự kiện dep(lan) trong phần clauses và do hai vị từ đồng nhất với nhau và hai đối số hằng đồng nhất nhau nên kết quả là Yes.

Khi dùng goal dep(X) thì dep sẽ được đồng nhất với dep và biến X đồng nhất với hằng lan, do đó ta có kết quả X=lan. Tương tự X=hong và X=binh.

Quay lui

Giả sử hệ thống đang chứng minh goal g, trong đó g được mô tả như sau:

g :- g1, g2, …, gj-1, gj, …, gn.

Khi các gi kiểm chứng từ trái sang phải, đến gj là sai thì hệ thống sẽ quay lui lại gj-1 để tìm lời giải khác.

Chẳng hạn trong ví dụ 2 nói trên, khi ta yêu cầu Goal: thich(lan,X), ta được X=binh.

Vị từ thich(lan,X) sẽ được đồng nhất với thich(lan,X) trong phần clauses, theo đó hệ thống phải chứng minh thich(lan,X):-khoe(X), dep(X), thong_minh(X).

- Trước hết đồng nhất khoe(X) với khoe(thuy) => X=thuy.
- Do dep(thuy) sai nên quay lui đồng nhất khoe(X) với khoe(lan) => X=lan.
- Do dep(lan) đúng nên tiếp tục kiểm tra thong_minh(lan).

- Do thong_minh(lan) sai nên quay lui để đồng nhất khoe(X) với khoe(binh) để có X=binh, sau đó kiểm tra thấy dep(binh) và thong_minh(binh) đều đúng nên X=binh là một nghiệm.

Bộ ký tự, từ khoá

Prolog dùng bộ ký tự sau: các chữ cái và chữ số (A – Z, a – z, 0 – 9); các toán tử (+, -, *, /, <, =, >) và các ký hiệu đặc biệt.

Một số từ khoá:

1. Trace: Khi có từ khoá này ở đầu chương trình, thì chương trình được thực hiện từng bước để theo dõi; dùng phím F10 để tiếp tục.
2. Fail: Khi ta dùng goal nội, chương trình chỉ cho ta một kết quả (mặc dù có thể còn những kết quả khác), để nhận về tất cả các kết quả khi chạy goal nội, ta dùng toán tử Fail.
3. ! hay còn gọi là nhát cắt, goal ngoại luôn cho ta mọi kết quả, muốn nhận chỉ một kết quả từ goal ngoại, ta dùng ký hiệu !.

Các kiểu dữ liệu

Trong prolog có kiểu dữ liệu chuẩn và kiểu do người lập trình định nghĩa.

Kiểu dữ liệu chuẩn

Là kiểu dữ liệu do prolog định nghĩa sẵn. Prolog cung cấp các kiểu dữ liệu chuẩn là: char, integer, real, string và symbol.

1. Char: Là kiểu ký tự. Hằng ký tự phải nằm giữa hai dấu nháy đơn.

Ví dụ: ‘a’, ‘#’.

1. Integer: Là kiểu số nguyên, tập giá trị bao gồm các số nguyên từ -32768 đến 32767.
2. Real: Là kiểu số thực, tập giá trị bao gồm các số thực thuộc hai đoạn: đoạn các số âm từ -10307 đến -10-307 và đoạn số dương từ 10-307 đến 10307.
3. String: Là kiểu chuỗi ký tự. Hằng chuỗi ký tự phải nằm giữa hai dấu nháy kép.

Ví dụ: “Turbo prolog 2.0”

1. Symbol: Là một kiểu sơ cấp, có hình thức giống chuỗi ký tự. Hằng symbol có hai dạng: Dãy các chữ, số và dấu gạch dưới viết liên tiếp, ký tự đầu phải viết thường (chẳng hạn: telephone_number); Dãy các ký tự ở giữa một cặp hai nháy kép (giống như chỗi ký tự)
2. Một số phép toán của các kiểu

Phép toán số học

Phép toán	Ý nghĩa	Kiểu của đối số	Kiểu kết quả
+	Cộng hai số	Integer, real	giống kiểu đối số
-	Trừ hai số	Integer, real	giống kiểu đối số
*	Nhân hai số	Integer, real	giống kiểu đối số
/	Chia hai số	Integer, real	giống kiểu đối số
Mod	Phép chia lấy phần dư	Integer	Integer
Div	Phép chia lấy phần nguyên	Integer	Integer

Phép toán quan hệ

Phép toán	Ý nghĩa	Kiểu của đối số	Kết quả

<	Nhỏ hơn	Char, integer, real, string	Yes hoặc No
<=	Nhỏ hơn hay bằng	Char, integer, real, string	Yes hoặc No
=	Bằng	Char, integer, real, string	Yes hoặc No
>	Lớn hơn	Char, integer, real, string	Yes hoặc No
>=	Lớn hơn hay bằng	Char, integer, real, string	Yes hoặc No
<> hay ><	Khác	Char, integer, real, string	Yes hoặc No

Các vị từ như các hàm toán học

Vị từ	Ý nghĩa	Kiểu của đối số	Kiểu kết quả	Ví dụ
Sin(X)	Tính sin của X	real	real	
Tan(X)	Tính tang của X	real	real	
Arctan(X)	Tính arctang của X	real	real	
Exp(X)	Tính eX	real	real	
Ln(X)	Tính logarit cơ số e của X	real	real	
Log(X)	Tính Logarit cơ số 10 của X	real	real	
SQRT(X)	Tính căn bậc hai của X	real	real	

ROUND(X)	Cho ta số nguyên là số X được làm tròn, dấu là dấu của X	real	integer	round(2.3)=2round(2.5)=3round(-2.5)=-2round(-2.6)=-3
TRUNC(X)	Cho phần nguyên của số X, dấu là dấu của X	real	integer	trunc(2.5)=2trunc(-2.6)=-2
ABS(X)	Cho ta trị tuyệt đối của X	real	real	
Random(X)	Cho ta số thực X nằm trong khoảng [0, 1)	real	real	
Random(Y, X)	Cho ta số nguyên X nằm trong khoảng [0, Y)	real	integer	

Toán tử NOT(X) : Nếu X là Yes thì NOT(X) là No và ngược lại.

Các kiểu dữ liệu do người lập trình định nghĩa

1. Kiểu mẩu tin:

Cú pháp: <tên kiểu mẩu tin> = tên mẩu tin (danh sách các kiểu phần tử)

Ví dụ:

Domains

ten, tac_gia, nha_xb, dia_chi = string

nam, thang, so_luong = integer

dien_tich = real

nam_xb = nxb(thang, nam)

do_vat = sach(tac_gia, ten, nha_xb, nam_xb); xe(ten, so_luong); nha(dia_chi, dien_tich)

predicates

so_huu(ten,do_vat)

clauses

so_huu(“Nguyen Van A”, sach(“Do Xuan Loi”, “Cau truc DL”, “Khoa hoc Ky thuat”, nxb(8,1985))).

so_huu(“Le thi B”, xe(“Dream II”, 2)).

so_huu(“Nguyen Huu C”, nha(“3/1 Ly Tu Trong, tp Can Tho”, 100.5))

1. Kiểu danh sách

Cú pháp: <tên kiểu danh sách> = <tên kiểu phần tử>*

Ví dụ:

Domains

intlist = integer*

Một danh sách là một dãy các phần tử phân cách nhau bởi dấu phẩy và đặt trong cặp dấu ngoặc vuông.

Ví dụ:

[]% Danh sách rỗng

[1,2,3] % Danh sách gồm ba số nguyên 1, 2 và 3.

Cấu trúc của danh sách bao gồm hai phần: Phần đầu là phần tử đầu tiên của danh sách và phần đuôi là một danh sách của các phần tử còn lại.

Danh sách được viết theo dạng [X|Y] thì X là phần tử đầu và Y là danh sách đuôi. Chẳng hạn trong danh sách [1,2,3] thì đầu là số nguyên 1 và đuôi là danh sách [2,3].

Trong danh sách cũng có thể dùng biến tự do, chẳng hạn ta có thể viết [_|Y] để chỉ một danh sách có đầu là một phần tử nào đó và có đuôi là danh sách Y.

Các hàm xuất nhập chuẩn

Xuất ra màn hình

1. Write(Arg1, Arg2, … ,Argn) in ra màn hình giá trị của các đối số.
2. Writef(Formatstring, Arg1, Arg2, … ,Argn) in ra màn hình giá trị của các đối số theo định dạng được chỉ định trong Formastring.

Trong đó Formastring là một chuỗi có thể là:

- “%d”: In số thập phân bình thường; đối số phải là char hoặc integer.
- “%c”: Đối số là một số integer, in ký tự có mã Ascci là đối số đó, chẳng hạn writef(“%c”,65) được A.
- “%e”: In số thực dưới dạng lũy thừa của 10.
- “%x”: In số Hexa; đối số phải là char hoặc integer.
- “%s”: In một chuỗi hoặc một symbol.

Nhập vào từ bàn phím

1. Readln(X): Nhập một chuỗi ký tự vào biến X.
2. ReadInt(X): Nhập một số nguyên vào biến X.
3. ReadReal(X): Nhập một số thực vào biến X.
4. ReadChar(X): Nhập vào một ký tự vào biến X.

Kỹ thuật đệ quy

Đệ quy là kỹ thuật lập trình được sử dụng trong nhiều ngôn ngữ. Trong Turbo Prolog ta sử dụng đệ quy khi một vị từ được định nghĩa nhờ vào chính vị từ đó.

Như đã nói trong chương lập trình hàm, trong chương trình đệ quy phải có ít nhất một trường hợp dừng và lời gọi đệ quy phải chứa yếu tố dẫn đến trường hợp dừng. Trong Prolog, trường hợp dừng được thể hiện bằng một sự kiện, yếu tố dẫn đến trường hợp dừng thể hiện bằng một biến, liên hệ với biến ban đầu bởi một công thức.

Ví dụ 1: Tính n giai thừa.

Predicates

Facto (integer, integer)

Clauses

Facto(0,1):- !.

Facto(N, FactN) :- N > 0, M = N – 1, facto(M, factM), factN = N*factM.

Ở ví dụ trên ta đã định nghĩa một vị từ dùng để tính giá trị giai thừa của một số tự nhiên, đối số thứ nhất là số cần tính giai thừa và đối số thứ hai dùng để nhận giá trị trả về.

Trường hợp dừng ở đây được xác đinh bởi sự kiện 0 giai thừa là 1.

Để tính N! ta tính M! với M= N-1. Yếu tố dẫn đến trường hợp dừng là biến M có giá trị bằng N-1.

Ví dụ 2: Xác định một phần tử trong danh sách các symbol

domains

symbol_list = symbol*

predicates

element1(integer,symbol_list,symbol)

element (integer,symbol_list,symbol)

clauses

% element1 không suy diễn ngược được

element1(1,[X|_],X).

element1(N,[_|L],Y):-M=N-1,

element1(M,L,Y).

% element có thể suy diễn ngược

element(1,[X|_],X).

element(N,[_|L],Y):-element(M,L,Y),

N=M+1.

Sự suy diễn thuận chiều là cho danh sách và vị trí, tìm được phần tử tại vị trí đó, chẳng hạn, nếu ta đưa vào goal element(2,[a,b,c,d],X) ta được X=b.

Sự suy diễn ngược ở đây là cho danh sách và phần tử, tìm được vị trí của phần tử đó, chẳng hạn, nếu ta đưa vào goal element(N,[a,b,c,d], b) ta được N=2.

Ví dụ 3: Sắp xếp một danh sách các số nguyên

domains

list=integer*

predicates

insert(integer,list,list)

sort(list,list)

clauses

insert(E,[],[E]).

insert(E,[A|B],[E,A|B]):-E<=A.

insert(E,[A|B],[A|C]):-E>A,insert(E,B,C).

sort([],[]).

sort([X|R1],L):-sort(R1,R),

insert(X,R,L).

Lập trình cấu trúc dữ liệu và giải thuật

CHƯƠNG 1 : THUẬT TOÁN – THUẬT GIẢI

I. KHÁI NIỆM THUẬT TOÁN – THUẬT GIẢI

II. THUẬT GIẢI HEURISTIC

III. CÁC PHƯƠNG PHÁP TÌM KIẾM HEURISTIC

III.1. Cấu trúc chung của bài toán tìm kiếm

III.2. Tìm kiếm chiều sâu và tìm kiếm chiều rộng

III.3. Tìm kiếm leo đồi

III.4. Tìm kiếm ưu tiên tối ưu (best-first search)

III.5. Thuật giải AT

III.6. Thuật giải AKT

III.7. Thuật giải A*

III.8. Ví dụ minh họa hoạt động của thuật giải A*

III.9. Bàn luận về A*

III.10. Ứng dụng A* để giải bài toán Ta-canh

III.11. Các chiến lược tìm kiếm lai

I. TỔNG QUAN THUẬT TOÁN – THUẬT GIẢI

Trong quá trình nghiên cứu giải quyết các vấn đề – bài toán, người ta đã đưa ra những nhận xét như sau:



Có nhiều bài toán cho đến nay vẫn chưa tìm ra một cách giải theo kiểu thuật toán và cũng không biết là có tồn tại thuật toán hay không.

- Có nhiều bài toán đã có thuật toán để giải nhưng không chấp nhận được vì thời gian giải theo thuật toán đó quá lớn hoặc các điều kiện cho thuật toán khó đáp ứng.

- Có những bài toán được giải theo những cách giải vi phạm thuật toán nhưng vẫn chấp nhận được.

Từ những nhận định trên, người ta thấy rằng cần phải có những đổi mới cho khái niệm thuật toán. Người ta đã mở rộng hai tiêu chuẩn của thuật toán: tính xác định và tính đúng đắn. Việc mở rộng tính xác định đối với thuật toán đã được thể hiện qua các giải thuật đệ quy và ngẫu nhiên. Tính đúng của thuật toán bây giờ không còn bắt buộc đối với một số cách giải bài toán, nhất là các cách giải gần đúng. Trong thực tiễn có nhiều trường hợp người ta chấp nhận các cách giải thường cho kết quả tốt (nhưng không phải lúc nào cũng tốt) nhưng ít phức tạp và hiệu quả. Chẳng hạn nếu giải một bài toán bằng thuật toán tối ưu đòi hỏi máy tính thực hiên nhiều năm thì chúng ta có thể sẵn lòng chấp nhận một giải pháp gần tối ưu mà chỉ cần máy tính chạy trong vài ngày hoặc vài giờ.

Các cách giải chấp nhận được nhưng không hoàn toàn đáp ứng đầy đủ các tiêu chuẩn của thuật toán thường được gọi là các thuật giải. Khái niệm mở rộng này của thuật toán đã mở cửa cho chúng ta trong việc tìm kiếm phương pháp để giải quyết các bài toán được đặt ra.

Một trong những thuật giải thường được đề cập đến và sử dụng trong khoa học trí tuệ nhân tạo là các cách giải theo kiểu Heuristic

II. THUẬT GIẢI HEURISTIC

Thuật giải Heuristic là một sự mở rộng khái niệm thuật toán. Nó thể hiện cách giải bài toán với các đặc tính sau:



Thường tìm được lời giải tốt (nhưng không chắc là lời giải tốt nhất)



Giải bài toán theo thuật giải Heuristic thường dễ dàng và nhanh chóng đưa ra kết quả hơn so với giải thuật tối ưu, vì vậy chi phí thấp hơn.



Thuật giải Heuristic thường thể hiện khá tự nhiên, gần gũi với cách suy nghĩ và hành động của con người.

Có nhiều phương pháp để xây dựng một thuật giải Heuristic, trong đó người ta thường dựa vào một số nguyên lý cơ bản như sau:



Nguyên lý vét cạn thông minh: Trong một bài toán tìm kiếm nào đó, khi không gian tìm kiếm lớn, ta thường tìm cách giới hạn lại không gian tìm kiếm hoặc thực hiện một kiểu dò tìm đặc biệt dựa vào đặc thù của bài toán để nhanh chóng tìm ra mục tiêu.



Nguyên lý tham lam (Greedy): Lấy tiêu chuẩn tối ưu (trên phạm vi toàn cục) của bài toán để làm tiêu chuẩn chọn lựa hành động cho phạm vi cục bộ của từng bước (hay từng giai đoạn) trong quá trình tìm kiếm lời giải.



Nguyên lý thứ tự: Thực hiện hành động dựa trên một cấu trúc thứ tự hợp lý của không gian khảo sát nhằm nhanh chóng đạt được một lời giải tốt.



Hàm Heuristic: Trong việc xây dựng các thuật giải Heuristic, người ta thường dùng các hàm Heuristic. Đó là các hàm đánh già thô, giá trị của hàm phụ thuộc vào trạng thái hiện tại của bài toán tại mỗi bước giải. Nhờ giá trị này, ta có thể chọn được cách hành động tương đối hợp lý trong từng bước của thuật giải.



Bài toán hành trình ngắn nhất – ứng dụng nguyên lý Greedy

Bài toán: Hãy tìm một hành trình cho một người giao hàng đi qua n điểm khác nhau, mỗi điểm đi qua một lần và trở về điểm xuất phát sao cho tổng chiều dài đoạn đường cần đi là ngắn nhất. Giả sử rằng có con đường nối trực tiếp từ giữa hai điểm bất kỳ.

Tất nhiên ta có thể giải bài toán này bằng cách liệt kê tất cả con đường có thể đi, tính chiều dài của mỗi con đường đó rồi tìm con đường có chiều dài ngắn nhất. Tuy nhiên, cách giải này lại có độ phức tạp 0(n!) (một hành trình là một hoán vị của n điểm, do đó, tổng số hành trình là số lượng hoán vị của một tập n phần tử là n!). Do đó, khi số đại lý tăng thì số con đường phải xét sẽ tăng lên rất nhanh.

Một cách giải đơn giản hơn nhiều và thường cho kết quả tương đối tốt là dùng một thuật giải Heuristic ứng dụng nguyên lý Greedy. Tư tưởng của thuật giải như sau:



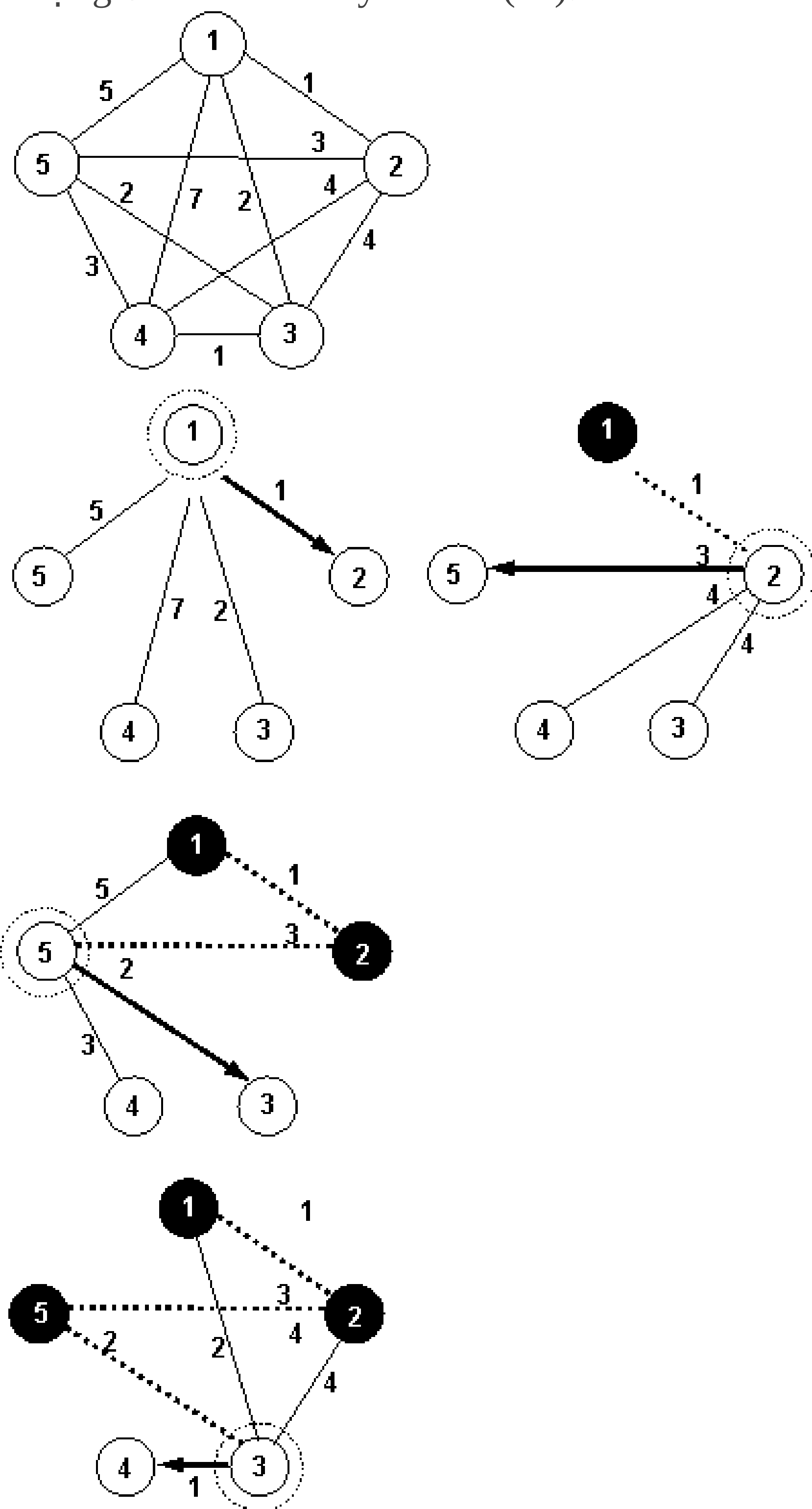
Từ điểm khởi đầu, ta liệt kê tất cả quãng đường từ điểm xuất phát cho đến n đại lý rồi chọn đi theo con đường ngắn nhất.



Khi đã đi đến một đại lý, chọn đi đến đại lý kế tiếp cũng theo nguyên tắc trên. Nghĩa là liệt kê tất cả con đường từ đại lý ta đang đứng đến những đại lý chưa đi đến. Chọn con đường ngắn nhất. Lặp lại quá trình này cho đến lúc không còn đại lý nào để đi.

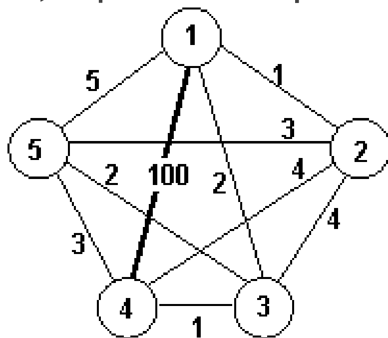
Bạn có thể quan sát hình sau để thấy được quá trình chọn lựa. Theo nguyên lý Greedy, ta lấy tiêu chuẩn hành trình ngắn nhất của bài toán làm tiêu chuẩn cho chọn lựa cục bộ. Ta hy vọng rằng, khi đi trên n đoạn đường ngắn nhất thì cuối cùng ta sẽ có một hành trình ngắn nhất. Điều này không phải lúc nào cũng đúng. Với điều kiện trong hình tiếp theo thì thuật giải cho chúng ta một hành trình có chiều dài là 14 trong khi hành trình tối ưu là 13. Kết quả của thuật giải Heuristic trong trường hợp này

chỉ lệch 1 đơn vị so với kết quả tối ưu. Trong khi đó, độ phức tạp của thuật giải Heuristic này chỉ là 0(n2).



Hình : Giải bài toán sử dụng nguyên lý Greedy

Tất nhiên, thuật giải theo kiểu Heuristic đôi lúc lại đưa ra kết quả không tốt, thậm chí rất tệ như trường hợp ở hình sau.

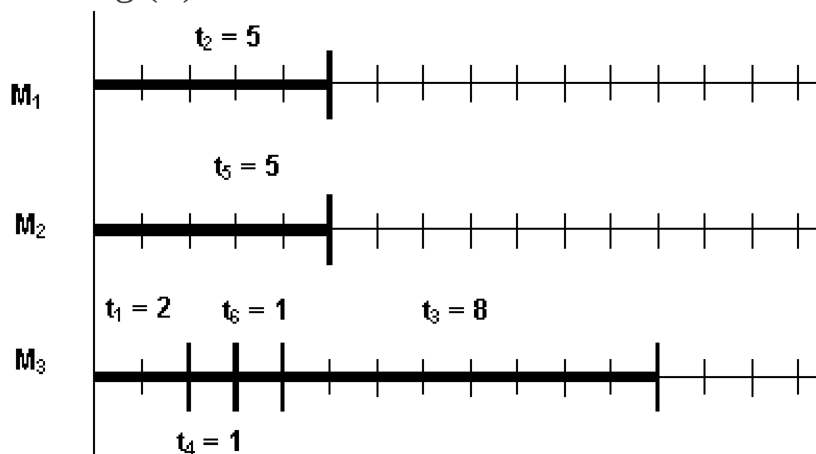




Bài toán phân việc – ứng dụng của nguyên lý thứ tự

Một công ty nhận được hợp đồng gia công m chi tiết máy J1, J2, … Jm. Công ty có n máy gia công lần lượt là P1, P2, … Pn. Mọi chi tiết đều có thể được gia công trên bất kỳ máy nào. Một khi đã gia công một chi tiết trên một máy, công việ sẽ tiếp tục cho đến lúc hoàn thành, không thể bị cắt ngang. Để gia công một việc J1 trên một máy bất kỳ ta cần dùng một thời gian tương ứng là t1. Nhiệm vụ của công ty là phải làm sao gia công xong toàn bộ n chi tiết trong thời gian sớm nhất.

Chúng ta xét bài toán trong trường hợp có 3 máy P1, P2, P3 và 6 công việc với thời gian là t1=2, t2=5, t3=8, t4=1, t5=5, t6=1. ta có một phương án phân công (L) như hình sau:



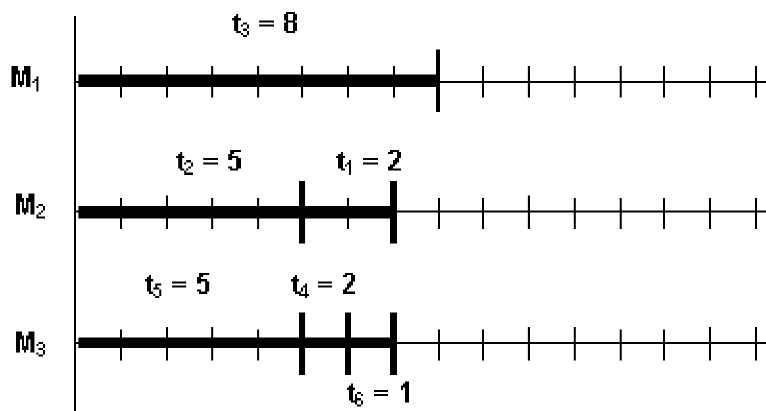
Theo hình này, tại thời điểm t=0, ta tiến hành gia công chi tiết J2 trên máy P1, J5 trên P2 và J1 tại P3. Tại thời điểm t=2, công việc J1 được hoàn thành, trên máy P3 ta gia công tiếp chi tiết J4. Trong lúc đó, hai máy P1 và P2 vẫn đang thực hiện công việc đầu tiên mình … Sơ đồ phân việc theo hình ở trên được gọi là lược đồ GANTT. Theo lược đồ này, ta thấy thời gian để hoàn thành toàn bộ 6 công việc là 12. Nhận xét một cách cảm tính ta thấy rằng phương án (L) vừa thực hiện là một phương án không tốt. Các máy P1 và P2 có quá nhiều thời gian rãnh.

Thuật toán tìm phương án tối ưu L0 cho bài toán này theo kiểu vét cạn có độ phức tạp cỡ O(mn) (với m là số máy và n là số công việc). Bây giờ ta xét đến một thuật giải Heuristic rất đơn giản (độ phức tạp O(n)) để giải bài toán này.

- Sắp xếp các công việc theo thứ tự giảm dần về thời gian gia công.

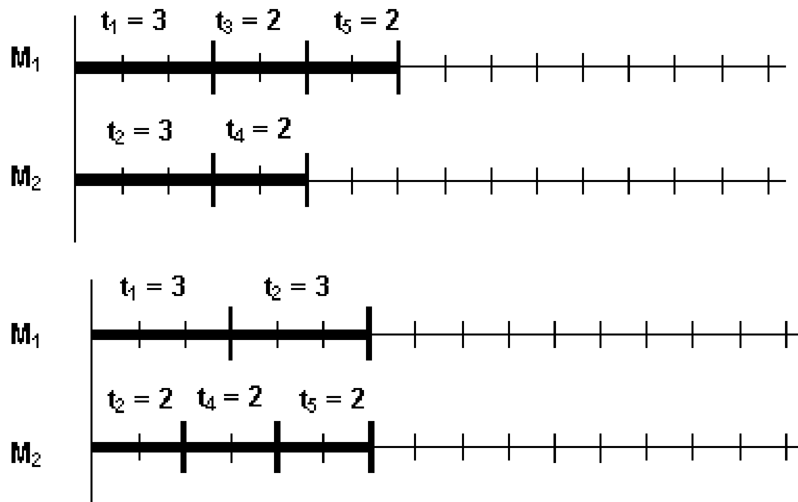
- Lần lượt sắp xếp các việc theo thứ tự đó vào máy còn dư nhiều thời gian nhất.

Với tư tưởng như vậy, ta sẽ có một phương án L* như sau:



Rõ ràng phương án L* vừa thực hiện cũng chính là phương án tối ưu của trường hợp này vì thời gian hoàn thành là 8, đúng bằng thời gian của công việc J3. Ta hy vọng rằng một giải Heuristic đơn giản như vậy sẽ là một

thuật giải tối ưu. Nhưng tiếc thay, ta dễ dàng đưa ra được một trường hợp mà thuật giải Heuristic không đưa ra được kết quả tối ưu.



Nếu gọi T* là thời gian để gia công xong n chi tiết máy do thuật giải Heuristic đưa ra và T0 là thời gian tối ưu thì người ta đã chứng minh được rằng

$$\frac{T^*}{T^0} \le \frac{4}{3} - \frac{1}{M}$$

, M là số máy

Với kết quả này, ta có thể xác lập được sai số mà chúng ta phải gánh chịu nếu dùng Heuristic thay vì tìm một lời giải tối ưu. Chẳng hạn với số máy là 2 (M=2) ta có

$$\frac{T^*}{T^0} \le \frac{7}{6}$$

, và đó chính là sai số cực đại mà trường hợp ở trên đã gánh chịu. Theo công thức này, số máy càng lớn thì sai số càng lớn.

Trong trường hợp M lớn thì tỷ số 1/M xem như bằng 0 . Như vậy, sai số tối đa mà ta phải chịu là T*   4/3 T0, nghĩa là sai số tối đa là 33%. Tuy

nhiên, khó tìm ra được những trường hợp mà sai số đúng bằng giá trị cực đại, dù trong trường hợp xấu nhất. Thuật giải Heuristic trong trường hợp này rõ ràng đã cho chúng ta những lời giải tương đối tốt.

III. CÁC PHƯƠNG PHÁP TÌM KIẾM HEURISTIC

Qua các phần trước chúng ta tìm hiểu tổng quan về ý tưởng của thuật giải Heuristic (nguyên lý Greedy và sắp thứ tự). Trong mục này, chúng ta sẽ đi sâu vào tìm hiểu một số kỹ thuật tìm kiếm Heuristic – một lớp bài toán rất quan trọng và có nhiều ứng dụng trong thực tế.

III.1. Cấu trúc chung của bài toán tìm kiếm

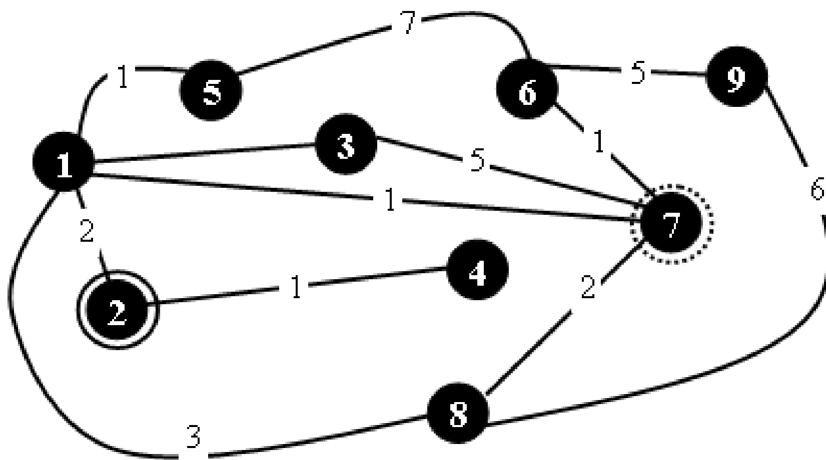
Để tiện lợi cho việc trình bày, ta hãy dành chút thời gian để làm rõ hơn "đối tượng" quan tâm của chúng ta trong mục này. Một cách chung nhất, nhiều vấn đề-bài toán phức tạp đều có dạng "tìm đường đi trong đồ thị" hay nói một cách hình thức hơn là "xuất phát từ một đỉnh của một đồ thị, tìm đường đi hiệu quả nhất đến một đỉnh nào đó". Một phát biểu khác thường gặp của dạng bài toán này là :

Cho trước hai trạng thái T0 và TG hãy xây dựng chuỗi trạng thái T0, T1, T2, ..., Tn-1, Tn = TG sao cho :

$$\sum_{1}^{n} \text{cost}(T_{i-1}, T_i)$$

thỏa mãn một điều kiện cho trước (thường là nhỏ nhất).

Trong đó, Ti thuộc tập hợp S (gọi là không gian trạng thái – state space) bao gồm tất cả các trạng thái có thể có của bài toán và cost(Ti-1, Ti) là chi phí để biến đổi từ trạng thái Ti-1 sang trạng thái Ti. Dĩ nhiên, từ một trạng thái Ti ta có nhiều cách để biến đổi sang trạng thái Ti+1. Khi nói đến một biến đổi cụ thể từ Ti-1 sang Ti ta sẽ dùng thuật ngữ hướng đi (với ngụ ý nói về sự lựa chọn).



Hình : Mô hình chung của các vấn đề-bài toán phải giải quyết bằng phương pháp tìm kiếm lời giải. Không gian tìm kiếm là một tập hợp trạng thái - tập các nút của đồ thị. Chi phí cần thiết để chuyển từ trạng thái T này sang trạng thái Tk được biểu diễn dưới dạng các con số nằm trên cung nối giữa hai nút tượng trưng cho hai trạng thái.

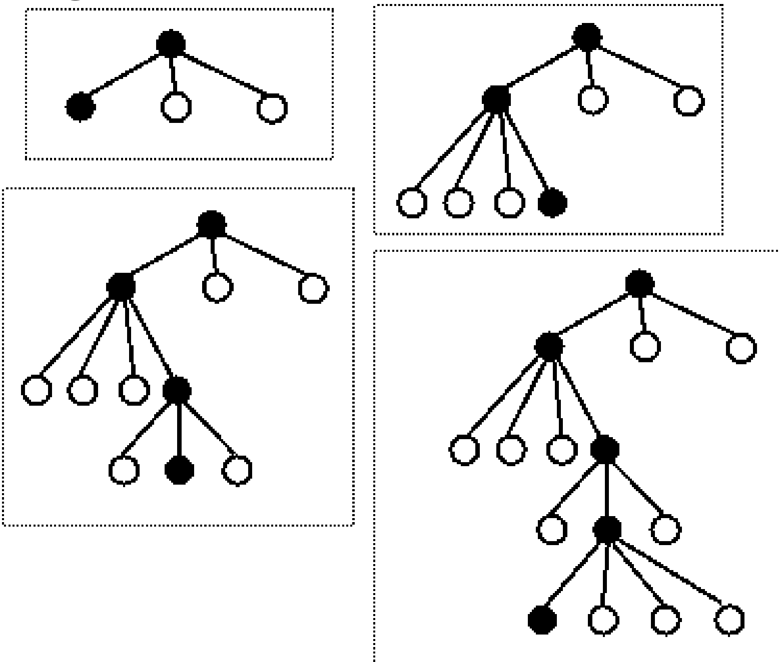
Đa số các bài toán thuộc dạng mà chúng ta đang mô tả đều có thể được biểu diễn dưới dạng đồ thị. Trong đó, một trạng thái là một đỉnh của đồ thị. Tập hợp S bao gồm tất cả các trạng thái chính là tập hợp bao gồm tất cả đỉnh của đồ thị. Việc biến đổi từ trạng thái Ti-1 sang trạng thái Ti là việc đi từ đỉnh đại diện cho Ti-1 sang đỉnh đại diện cho Ti theo cung nối giữa hai đỉnh này.

III.2. Tìm kiếm chiều sâu và tìm kiếm chiều rộng

Để bạn đọc có thể hình dung một cách cụ thể bản chất của thuật giải Heuristic, chúng ta nhất thiết phải nắm vững hai chiến lược tìm kiếm cơ bản là tìm kiếm theo chiều sâu (Depth First Search) và tìm kiếm theo chiều rộng (Breath First Search). Sở dĩ chúng ta dùng từ chiến lược mà không phải là phương pháp là bởi vì trong thực tế, người ta hầu như chẳng bao giờ vận dụng một trong hai kiểm tìm kiếm này một cách trực tiếp mà không phải sửa đổi gì.

III.2.1. Tìm kiếm chiều sâu (Depth-First Search)

Trong tìm kiếm theo chiều sâu, tại trạng thái (đỉnh) hiện hành, ta chọn một trạng thái kế tiếp (trong tập các trạng thái có thể biến đổi thành từ trạng thái hiện tại) làm trạng thái hiện hành cho đến lúc trạng thái hiện hành là trạng thái đích. Trong trường hợp tại trạng thái hiện hành, ta không thể biến đổi thành trạng thái kế tiếp thì ta sẽ quay lui (back-tracking) lại trạng thái trước trạng thái hiện hành (trạng thái biến đổi thành trạng thái hiện hành) để chọn đường khác. Nếu ở trạng thái trước này mà cũng không thể biến đổi được nữa thì ta quay lui lại trạng thái trước nữa và cứ thế. Nếu đã quay lui đến trạng thái khởi đầu mà vẫn thất bại thì kết luận là không có lời giải. Hình ảnh sau minh họa hoạt động của tìm kiếm theo chiều sâu.

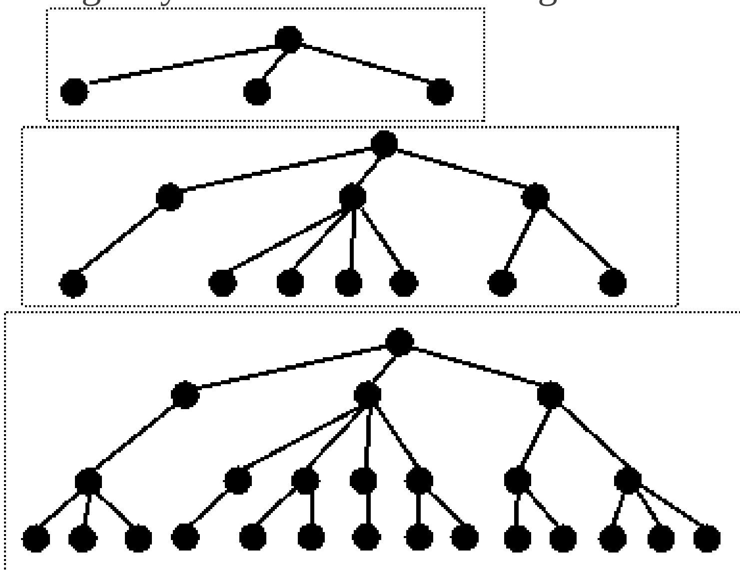


Hình : Hình ảnh của tìm kiếm chiều sâu. Nó chỉ lưu ý "mở rộng" trạng thái được chọn mà không "mở rộng" các trạng thái khác (nút màu trắng trong hình vẽ).

III.2.2. Tìm kiếm chiều rộng (Breath-First Search)

Ngược lại với tìm kiếm theo kiểu chiều sâu, tìm kiếm chiều rộng mang hình ảnh của vết dầu loang. Từ trạng thái ban đầu, ta xây dựng tập hợp S bao gồm các trạng thái kế tiếp (mà từ trạng thái ban đầu có thể biến đổi thành). Sau đó, ứng với mỗi trạng thái Tk trong tập S, ta xây dựng tập Sk

bao gồm các trạng thái kế tiếp của Tk rồi lần lượt bổ sung các Sk vào S. Quá trình này cứ lặp lại cho đến lúc S có chứa trạng thái kết thúc hoặc S không thay đổi sau khi đã bổ sung tất cả Sk.



Hình : Hình ảnh của tìm kiếm chiều rộng. Tại một bước, mọi trạng thái đều được mở rộng, không bỏ sót trạng thái nào.

	Chiều sâu	Chiều rộng
Tính hiệu quả	Hiệu quả khi lời giải nằm sâu trong cây tìm kiếm và có một phương án chọn hướng đi chính xác. Hiệu quả của chiến lược phụ thuộc vào phương án chọn hướng đi. Phương án càng kém hiệu quả thì hiệu quả của chiến	Hiệu quả khi lời giải nằm gần gốc của cây tìm kiếm. Hiệu quả của chiến lược phụ thuộc vào độ sâu của lời giải. Lời giải càng xa gốc thì hiệu quả của chiến

	lược càng giảm. Thuận lợi khi muốn tìm chỉ một lời giải.	lược càng giảm. Thuận lợi khi muốn tìm nhiều lời giải.
Lượng bộ nhớ sử dụng để lưu trữ các trạng thái	Chỉ lưu lại các trạng thái chưa xét đến.	Phải lưu toàn bộ các trạng thái.
Trường hợp xấu nhất	Vét cạn toàn bộ	Vét cạn toàn bộ.
Trường hợp tốt nhất	Phương án chọn hướng đi tuyệt đối chính xác. Lời giải được xác định một cách trực tiếp.	Vét cạn toàn bộ.

Tìm kiếm chiều sâu và tìm kiếm chiều rộng đều là các phương pháp tìm kiếm có hệ thống và chắc chắn tìm ra lời giải. Tuy nhiên, do bản chất là vét cạn nên với những bài toán có không gian lớn thì ta không thể dùng hai chiến lược này được. Hơn nữa, hai chiến lược này đều có tính chất "mù quáng" vì chúng không chú ý đến những thông tin (tri thức) ở trạng thái hiện thời và thông tin về đích cần đạt tới cùng mối quan hệ giữa chúng. Các tri thức này vô cùng quan trọng và rất có ý nghĩa để thiết kế các thuật giải hiệu quả hơn mà ta sắp sửa bàn đến.

III.3. Tìm kiếm leo đồi

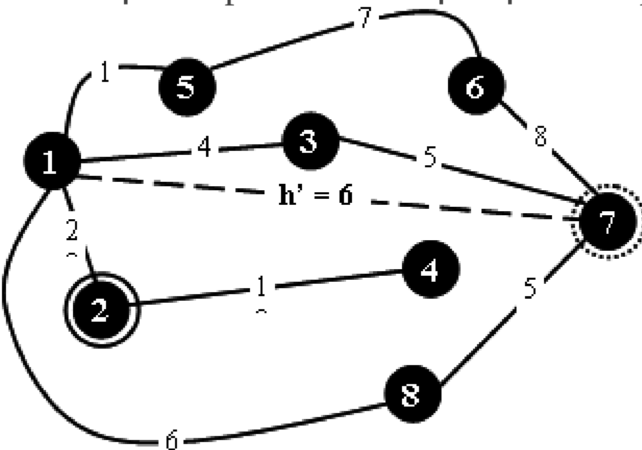
III.3.1. Leo đồi đơn giản

Tìm kiếm leo đồi theo đúng nghĩa, nói chung, thực chất chỉ là một trường hợp đặc biệt của tìm kiếm theo chiều sâu nhưng không thể quay lui. Trong tìm kiếm leo đồi, việc lựa chọn trạng thái tiếp theo được quyết định dựa trên một hàm Heuristic.



Hàm Heuristic là gì ?

Thuật ngữ "hàm Heuristic" muốn nói lên điều gì? Chẳng có gì ghê gớm. Bạn đã quen với nó rồi! Đó đơn giản chỉ là một ước lượng về khả năng dẫn đến lời giải tính từ trạng thái đó (khoảng cách giữa trạng thái hiện tại và trạng thái đích). Ta sẽ quy ước gọi hàm này là h trong suốt giáo trình này. Đôi lúc ta cũng đề cập đến chi phí tối ưu thực sự từ một trạng thái dẫn đến lời giải. Thông thường, giá trị này là không thể tính toán được (vì tính được đồng nghĩa là đã biết con đường đến lời giải !) mà ta chỉ dùng nó như một cơ sở để suy luận về mặt lý thuyết mà thôi ! Hàm h, ta quy ước rằng, luôn trả ra kết quả là một số không âm. Để bạn đọc thực sự nắm được ý nghĩa của hai hàm này, hãy quan sát hình sau trong đó minh họa chi phí tối ưu thực sự và chi phí ước lượng.



Hình Chi phí ước lượng h’ = 6 và chi phí tối ưu thực sự h = 4+5 = 9 (đi theo đường 1-3-7)

Bạn đang ở trong một thành phố xa lạ mà không có bản đồ trong tay và ta muốn đi vào khu trung tâm? Một cách suy nghĩ đơn giản, chúng ta sẽ nhắm vào hướng những tòa cao ốc của khu trung tâm!



Tư tưởng

1) Nếu trạng thái bắt đầu cũng là trạng thái đích thì thoát và báo là đã tìm được lời giải. Ngược lại, đặt trạng thái hiện hành (Ti) là trạng thái khởi đầu (T0)

2) Lặp lại cho đến khi đạt đến trạng thái kết thúc hoặc cho đến khi không tồn tại một trạng thái tiếp theo hợp lệ (Tk) của trạng thái hiện hành :

a. Đặt Tk là một trạng thái tiếp theo hợp lệ của trạng thái hiện hành Ti.

b. Đánh giá trạng thái Tk mới :

b.1. Nếu là trạng thái kết thúc thì trả về trị này và thoát.

b.2. Nếu không phải là trạng thái kết thúc nhưng tốt hơn trạng thái hiện hành thì cập nhật nó thành trạng thái hiện hành.

b.3. Nếu nó không tốt hơn trạng thái hiện hành thì tiếp tục vòng lặp.



Mã giả

Ti := T0; Stop :=FALSE;

WHILE Stop=FALSE DO BEGIN

IF Ti   TG THEN BEGIN

<tìm được kết quả >; Stop:=TRUE;

END;

ELSE BEGIN

Better:=FALSE;

```
WHILE (Better=FALSE) AND (STOP=FALSE) DO BEGIN

IF <không tồn tại trạng thái kế tiếp hợp lệ của Ti> THEN BEGIN

<không tìm được kết quả >; Stop:=TRUE; END;

ELSE BEGIN

Tk := <một trạng thái kế tiếp hợp lệ của Ti>;

IF <h(Tk) tốt hơn h(Ti)> THEN BEGIN

Ti :=Tk; Better:=TRUE;

END;

END;

END; {WHILE}

END; {ELSE}

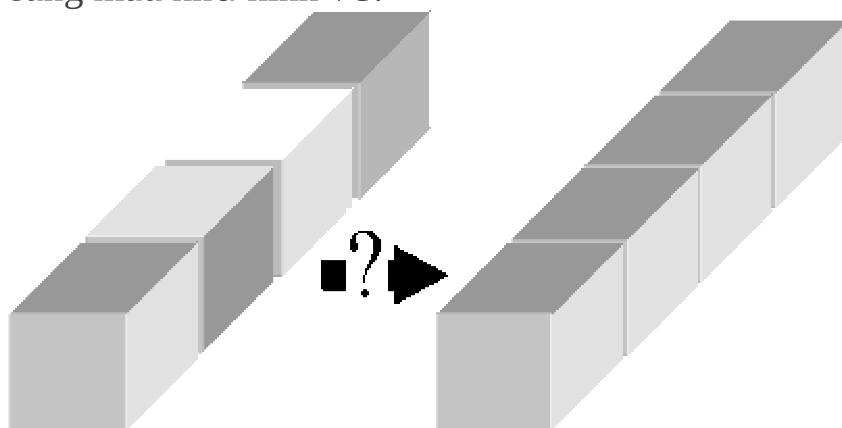
END;{WHILE}
```

Mệnh đề "h’(Tk) tốt hơn h’(Ti)" nghĩa là gì? Đây là một khái niệm chung chung. Khi cài đặt thuật giải, ta phải cung cấp một định nghĩa tường minh về tốt hơn. Trong một số trường hợp, tốt hơn là nhỏ hơn : h’(Tk) < h’(Ti); một số trường hợp khác tốt hơn là lớn hơn h’(Tk) > h’(Ti)...Chẳng hạn, đối với bài toán tìm đường đi ngắn nhất giữa hai điểm. Nếu dùng hàm h’ là hàm cho ra khoảng cách theo đường chim bay giữa vị trí hiện tại (trạng thái hiện tại) và đích đến (trạng thái đích) thì tốt hơn nghĩa là nhỏ hơn.

Vấn đề cần làm rõ kế tiếp là thế nào là <một trạng thái kế tiếp hợp lệ của Ti>? Một trạng thái kế tiếp hợp lệ là trạng thái chưa được xét đến. Giả sử h của trạng thái hiện tại Ti có giá trị là h(Ti) = 1.23 và từ Ti ta có thể biến đổi sang một trong 3 trạng thái kế tiếp lần lượt là Tk1, Tk2, Tk3 với giá trị các hàm h tương ứng là h(Tk1) = 1.67, h(Tk2) = 2.52,

h’(Tk3) = 1.04. Đầu tiên, Tk sẽ được gán bằng Tk1, nhưng vì h’(Tk) = h’(Tk1) > h’(Ti) nên Tk không được chọn. Kế tiếp là Tk sẽ được gán bằng Tk2 và cũng không được chọn. Cuối cùng thì Tk3 được chọn. Nhưng giả sử h’(Tk3) = 1.3 thì cả Tk3 cũng không được chọn và mệnh đề <không thể sinh ra trạng thái kế tiếp của Ti> sẽ có giá trị TRUE. Giải thích này có vẻ hiển nhiên nhưng có lẽ cần thiết để tránh nhầm lẫn cho bạn đọc.

Để thấy rõ hoạt động của thuật giải leo đồi. Ta hãy xét một bài toán minh họa sau. Cho 4 khối lập phương giống nhau A, B, C, D. Trong đó các mặt (M1), (M2), (M3), (M4), (M5), (M6) có thể được tô bằng 1 trong 6 màu (1), (2), (3), (4), (5), (6). Ban đầu các khối lập phương được xếp vào một hàng. Mỗi một bước, ta chỉ được xoay một khối lập phương quanh một trục (X,Y,Z) 900 theo chiều bất kỳ (nghĩa là ngược chiều hay thuận chiều kim đồng hồ cũng được). Hãy xác định số bước quay ít nhất sao cho tất cả các mặt của khối lập phương trên 4 mặt của hàng là có cùng màu như hình vẽ.



Hình : Bài toán 4 khối lập phương

Để giải quyết vấn đề, trước hết ta cần định nghĩa một hàm G dùng để đánh giá một tình trạng cụ thể có phải là lời giải hay không? Bạn đọc có thể dễ dàng đưa ra một cài đặt của hàm G như sau :

IF (Gtrái + Gphải + Gtrên + Gdưới + Gtrước + Gsau) = 16 THEN

G:=TRUE

ELSE

G:=FALSE;

Trong đó, Gphải là số lượng các mặt có cùng màu của mặt bên phải của hàng. Tương tự cho Gtrái, Gtrên, Ggiữa, Gtrước, Gsau. Tuy nhiên, do các khối lập phương A,B,C,D là hoàn toàn tương tự nhau nên tương quan giữa các mặt của mỗi khối là giống nhau. Do đó, nếu có 2 mặt không đối nhau trên hàng đồng màu thì 4 mặt còn lại của hàng cũng đồng màu. Từ đó ta chỉ cần hàm G được định nghĩa như sau là đủ :

IF Gphải + Gdưới = 8 THEN

G:=TRUE

ELSE

G:=FALSE;

Hàm h (ước lượng khả năng dẫn đến lời giải của một trạng thái) sẽ được định nghĩa như sau :

h = Gtrái + Gphải + Gtrên + Gdưới

Bài toán này đủ đơn giản để thuật giải leo đồi có thể hoạt động tốt. Tuy nhiên, không phải lúc nào ta cũng may mắn như thế!

Đến đây, có thể chúng ta sẽ nảy sinh một ý tưởng. Nếu đã chọn trạng thái tốt hơn làm trạng thái hiện tại thì tại sao không chọn trạng thái tốt nhất ? Như vậy, có lẽ ta sẽ nhanh chóng dẫn đến lời giải hơn! Ta sẽ bàn luận về vấn đề: "liệu cải tiến này có thực sự giúp chúng ta dẫn đến lời giải nhanh hơn hay không?" ngay sau khi trình bày xong thuật giải leo đồi dốc đứng.

III.3.2. Leo đồi dốc đứng

Về cơ bản, leo đồi dốc đứng cũng giống như leo đồi, chỉ khác ở điểm là leo đồi dốc đứng sẽ duyệt tất cả các hướng đi có thể và chọn đi theo trạng thái tốt nhất trong số các trạng thái kế tiếp có thể có (trong khi đó leo đồi chỉ chọn đi theo trạng thái kế tiếp đầu tiên tốt hơn trạng thái hiện hành mà nó tìm thấy).



Tư tưởng

1) Nếu trạng thái bắt đầu cũng là trạng thái đích thì thoát và báo là đã tìm được lời giải. Ngược lại, đặt trạng thái hiện hành (Ti) là trạng thái khởi đầu (T0)

2) Lặp lại cho đến khi đạt đến trạng thái kết thúc hoặc cho đến khi (Ti) không tồn tại một trạng thái kế tiếp (Tk) nào tốt hơn trạng thái hiện tại (Ti)

a) Đặt S bằng tập tất cả trạng thái kế tiếp có thể có của Ti và tốt hơn Ti.

b) Xác định Tkmax là trạng thái tốt nhất trong tập S

Đặt Ti = Tkmax



Mã giả

Ti := T0;

Stop :=FALSE;

WHILE Stop=FALSE DO BEGIN

IF Ti   TG THEN BEGIN

<tìm được kết quả >;

STOP :=TRUE;

```
END;
ELSE BEGIN
Best:=h’(Ti);
Tmax := Ti;
WHILE <tồn tại trạng thái kế tiếp hợp lệ của Ti> DO BEGIN
Tk := <một trạng thái kế tiếp hợp lệ của Ti>;
IF <h’(Tk) tốt hơn Best> THEN BEGIN
Best :=h’(Tk);
Tmax := Tk;
END;
END;
IF (Best>Ti) THEN
Ti := Tmax;
ELSE BEGIN
<không tìm được kết quả >;
STOP:=TRUE;
END;
END; {ELSE IF}
END;{WHILE STOP}
```

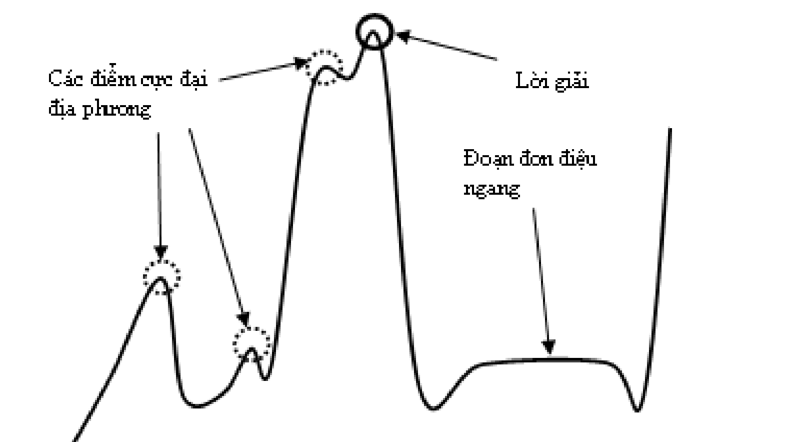
III.3.3. Đánh giá

So với leo đồi đơn giản, leo đồi dốc đứng có ưu điểm là luôn luôn chọn hướng có triển vọng nhất để đi. Liệu điều này có đảm bảo leo đồi dốc đứng luôn tốt hơn leo đồi đơn giản không? Câu trả lời là không. Leo đồi dốc đứng chỉ tốt hơn leo đồi đơn giản trong một số trường hợp mà thôi. Để chọn ra được hướng đi tốt nhất, leo đồi dốc đứng phải duyệt qua tất cả các hướng đi có thể có tại trạng thái hiện hành. Trong khi đó, leo đồi đơn giản chỉ chọn đi theo trạng thái đầu tiên tốt hơn (so với trạng thái hiện hành) mà nó tìm ra được. Do đó, thời gian cần thiết để leo đồi dốc đứng chọn được một hướng đi sẽ lớn hơn so với leo đồi đơn giản. Tuy vậy, do lúc nào cũng chọn hướng đi tốt nhất nên leo đồi dốc đứng thường sẽ tìm đến lời giải sau một số bước ít hơn so với leo đồi đơn giản. Nói một cách ngắn gọn, leo đồi dốc đứng sẽ tốn nhiều thời gian hơn cho một bước nhưng lại đi ít bước hơn; còn leo đồi đơn giản tốn ít thời gian hơn cho một bước đi nhưng lại phải đi nhiều bước hơn. Đây chính là yếu tố được và mất giữa hai thuật giải nên ta phải cân nhắc kỹ lưỡng khi lựa chọn thuật giải.

Cả hai phương pháp leo núi đơn giản và leo núi dốc đứng đều có khả năng thất bại trong việc tìm lời giải của bài toán mặc dù lời giải đó thực sự hiện hữu. Cả hai giải thuật đều có thể kết thúc khi đạt được một trạng thái mà không còn trạng thái nào tốt hơn nữa có thể phát sinh nhưng trạng thái này không phải là trạng thái đích. Điều này sẽ xảy ra nếu chương trình đạt đến một điểm cực đại địa phương, một đoạn đơn điệu ngang.

Điểm cực đại địa phương (a local maximum) : là một trạng thái tốt hơn tất cả lân cận của nó nhưng không tốt hơn một số trạng thái khác ở xa hơn. Nghĩa là tại một điểm cực đại địa phương, mọi trạng thái trong một lân cận của trạng thái hiện tại đều xấu hơn trạng thái hiện tại. Tuy có dáng vẻ của lời giải nhưng các cực đại địa phương không phải là lời giải thực sự. Trong trường hợp này, chúng được gọi là những ngọn đồi thấp.

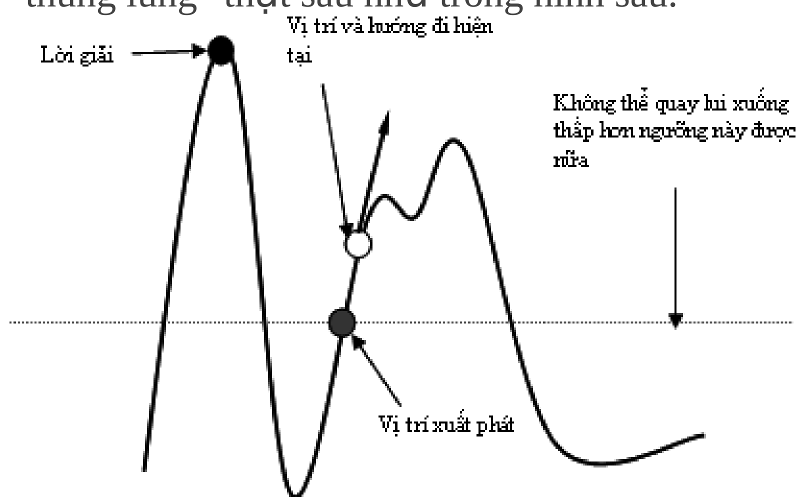
Đoạn đơn điệu ngang (a plateau) : là một vùng bằng phẳng của không gian tìm kiếm, trong đó, toàn bộ các trạng thái lân cận đều có cùng giá trị.



Hình : Các tình huống khó khăn cho tìm kiếm leo đèo.

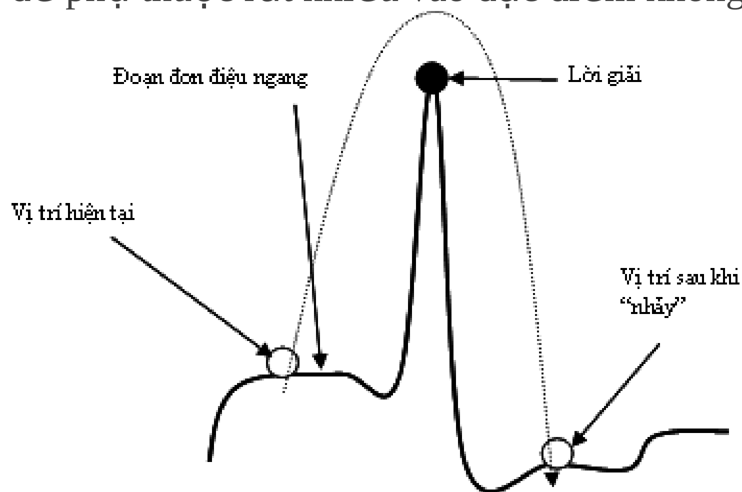
Để đối phó với các các điểm này, người ta đã đưa ra một số giải pháp. Ta sẽ tìm hiểu 2 trong số các giải pháp này. Những giải này, không thực sự giải quyết trọn vẹn vấn đề mà chỉ là một phương án cứu nguy tạm thời mà thôi.

Phương án đầu tiên là kết hợp leo đồi và quay lui. Ta sẽ quay lui lại các trạng thái trước đó và thử đi theo hướng khác. Thao tác này hợp lý nếu tại các trạng thái trước đó có một hướng đi tốt mà ta đã bỏ qua trước đó. Đây là một cách khá hay để đối phó với các điểm cực đại địa phương. Tuy nhiên, do đặc điểm của leo đồi là "bước sau cao hơn bước trước" nên phương án này sẽ thất bại khi ta xuất phát từ một điểm quá cao hoặc xuất phát từ một đỉnh đồi mà để đến được lời giải cần phải đi qua một "thung lũng" thật sâu như trong hình sau.



Hình : Một trường hợp thất bại của leo đèo kết hợp quay lui.

Cách thứ hai là thực hiện một bước nhảy vọt theo hướng nào đó để thử đến một vùng mới của không gian tìm kiếm. Nôm na là "bước" liên tục nhiều "bước" (chẳng hạn 5,7,10, …) mà tạm thời "quên" đi việc kiểm tra "bước sau cao hơn bước trước". Tiếp cận có vẻ hiệu quả khi ta gặp phải một đoạn đơn điệu ngang. Tuy nhiên, nhảy vọt cũng có nghĩa là ta đã bỏ qua cơ hội để tiến đến lời giải thực sự. Trong trường hợp chúng ta đang đứng khá gần lời giải, việc nhảy vọt sẽ đưa chúng ta sang một vị trí hoàn toàn xa lạ, mà từ đó, có thể sẽ dẫn chúng ta đến một rắc rối kiểu khác. Hơn nữa, số bước nhảy là bao nhiêu và nhảy theo hướng nào là một vấn đề phụ thuộc rất nhiều vào đặc điểm không gian tìm kiếm của bài toán.



Hình Một trường hợp khó khăn cho phương án "nhảy vọt".

Leo núi là một phương pháp cục bộ bởi vì nó quyết định sẽ làm gì tiếp theo dựa vào một đánh giá về trạng thái hiện tại và các trạng thái kế tiếp có thể có (tốt hơn trạng thái hiện tại, trạng thái tốt nhất tốt hơn trạng thái hiện tại) thay vì phải xem xét một cách toàn diện trên tất cả các trạng thái đã đi qua. Thuận lợi của leo núi là ít gặp sự bùng nổ tổ hợp hơn so với các phương pháp toàn cục. Nhưng nó cũng giống như các phương pháp cục bộ khác ở chỗ là không chắc chắn tìm ra lời giải trong trường hợp xấu nhất.

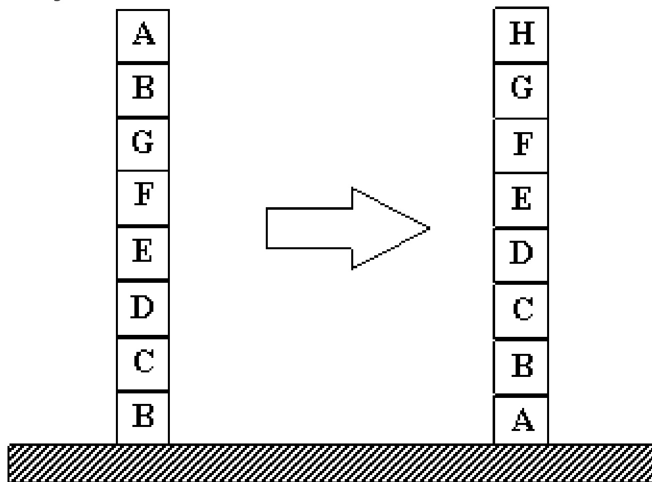
Một lần nữa, ta khẳng định lại vai trò quyết định của hàm Heuristic trong quá trình tìm kiếm lời giải. Với cùng một thuật giải (như leo đồi chẳng

hạn), nếu ta có một hàm Heuristic tốt hơn thì kết quả sẽ được tìm thấy nhanh hơn. Ta hãy xét bài toán về các khối được trình bày ở hình sau. Ta có hai thao tác biến đổi là:

+ Lấy một khối ở đỉnh một cột bất kỳ và đặt nó lên một chỗ trống tạo thành một cột mới. Lưu ý là chỉ có thể tạo ra tối đa 2 cột mới.

+ Lấy một khối ở đỉnh một cột và đặt nó lên đỉnh một cột khác

Hãy xác định số thao tác ít nhất để biến đổi cột đã cho thành cột kết quả.



Hình : Trạng thái khởi đầu và trạng thái kết thúc

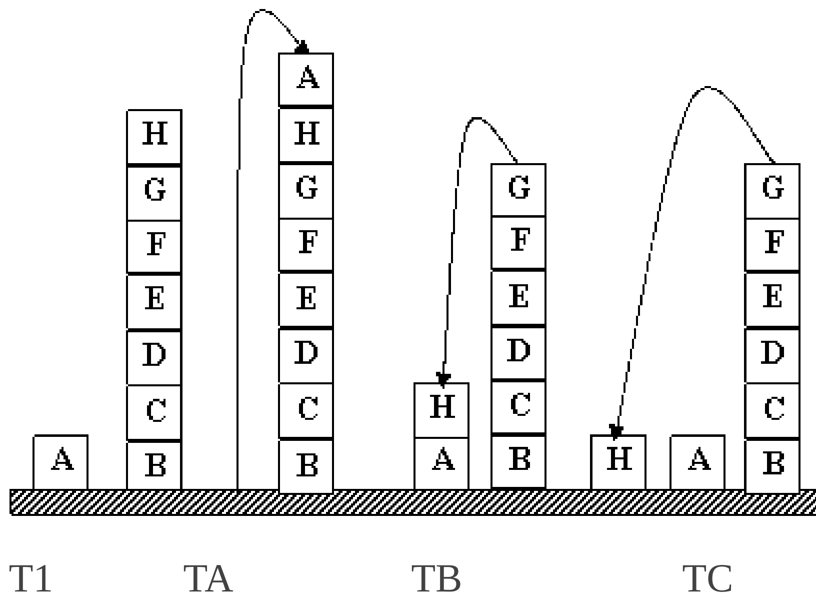
Giả sử ban đầu ta dùng một hàm Heuristic đơn giản như sau :

H1 : Cộng 1 điểm cho mỗi khối ở vị trí đúng so với trạng thái đích. Trừ 1 điểm cho mỗi khối đặt ở vị trí sai so với trạng thái đích.

Dùng hàm này, trạng thái kết thúc sẽ có giá trị là 8 vì cả 8 khối đều được đặt ở vị trí đúng. Trạng thái khởi đầu có giá trị là 4 (vì nó có 1 điểm cộng cho các khối C, D, E, F, G, H và 1 điểm trừ cho các khối A và B). Chỉ có thể có một di chuyển từ trạng thái khởi đầu, đó là dịch chuyển khối A xuống tạo thành một cột mới (T1).

Điều đó sinh ra một trạng thái với số điểm là 6 (vì vị trí của khối A bây giờ sinh ra 1 điểm cộng hơn là một điểm trừ). Thủ tục leo núi sẽ chấp nhận sự dịch chuyển đó. Từ trạng thái mới T1, có ba di chuyển có thể

thực hiện dẫn đến ba trạng thái Ta, Tb, Tc được minh họa trong hình dưới. Những trạng thái này có số điểm là : h’(Ta)= 4; h’(Tb) = 4 và h’(Tc) = 4



Hình Các trạng thái có thể đạt được từ T1

Thủ tục leo núi sẽ tạm dừng bởi vì tất cả các trạng thái này có số điểm thấp hơn trạng thái hiện hành. Quá trình tìm kiếm chỉ dừng lại ở một trạng thái cực đại địa phương mà không phải là cực đại toàn cục.

Chúng ta có thể đổ lỗi cho chính giải thuật leo đồi vì đã thất bại do không đủ tầm nhìn tổng quát để tìm ra lời giải. Nhưng chúng ta cũng có thể đổ lỗi cho hàm Heuristic và cố gắng sửa đổi nó. Giả sử ta thay hàm ban đầu bằng hàm Heuristic sau đây :

H2 : Đối với mỗi khối phụ trợ đúng (khối phụ trợ là khối nằm bên dưới khối hiện tại), cộng 1 điểm, ngược lại trừ 1 điểm.

Dùng hàm này, trạng thái kết thúc có số điểm là 28 vì B nằm đúng vị trí và không có khối phụ trợ nào, C đúng vị trí được 1 điểm cộng với 1 điểm do khối phụ trợ B nằm đúng vị trí nên C được 2 điểm, D được 3 điểm,

....Trạng thái khởi đầu có số điểm là –28. Việc di chuyển A xuống tạo thành một cột mới làm sinh ra một trạng thái với số điểm là h’(T1) = –21 vì A không còn 7 khối sai phía dưới nó nữa. Ba trạng thái có thể phát sinh tiếp theo bây giờ có các điểm số là : h’(Ta)=–28; h’(Tb)=–16 và h’(Tc) = –15. Lúc này thủ tục leo núi dốc đứng sẽ chọn di chuyến đến trạng thái Tc, ở đó có một khối đúng. Qua hàm H2 này ta rút ra một nguyên tắc : tốt hơn không chỉ có nghĩa là có nhiều ưu điểm hơn mà còn phải ít khuyết điểm hơn. Hơn nữa, khuyết điểm không có nghĩa chỉ là sự sai biệt ngay tại một vị trí mà còn là sự khác biệt trong tương quan giữa các vị trí. Rõ ràng là đứng về mặt kết quả, cùng một thủ tục leo đồi nhưng hàm H1 bị thất bại (do chỉ biết đánh giá ưu điểm) còn hàm H2 mới này lại hoạt động một cách hoàn hảo (do biết đánh giá cả ưu điểm và khuyết điểm).

Đáng tiếc, không phải lúc nào chúng ta cũng thiết kế được một hàm Heuristic hoàn hảo như thế. Vì việc đánh giá ưu điểm đã khó, việc đánh giá khuyết điểm càng khó và tinh tế hơn. Chẳng hạn, xét lại vấn đề muốn đi vào khu trung tâm của một thành phố xa lạ. Để hàm Heuristic hiệu quả, ta cần phải đưa các thông tin về các đường một chiều và các ngõ cụt, mà trong trường hợp một thành phố hoàn toàn xa lạ thì ta khó hoặc không thể biết được những thông tin này.

Đến đây, chúng ta hiểu rõ bản chất của hai thuật giải tiếp cận theo chiến lược tìm kiếm chiều sâu. Hiệu quả của cả hai thuật giải leo đồi đơn giản và leo đồi dốc đứng phụ thuộc vào :

+ Chất lượng của hàm Heuristic.

+ Đặc điểm của không gian trạng thái.

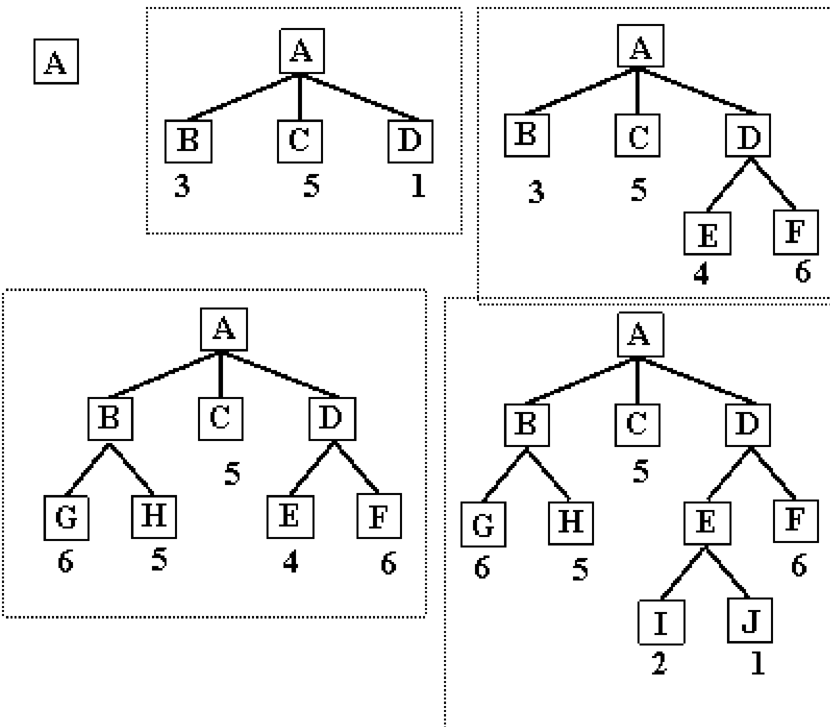
+ Trạng thái khởi đầu.

Sau đây, chúng ta sẽ tìm hiểu một tiếp cận theo mới, kết hợp được sức mạnh của cả tìm kiếm chiều sâu và tìm kiếm chiều rộng. Một thuật giải rất linh động và có thể nói là một thuật giải kinh điển của Heuristic.

III.4. Tìm kiếm ưu tiên tối ưu (best-first search)

Ưu điểm của tìm kiếm theo chiều sâu là không phải quan tâm đến sự mở rộng của tất cả các nhánh. Ưu điểm của tìm kiếm chiều rộng là không bị sa vào các đường dẫn bế tắc (các nhánh cụt). Tìm kiếm ưu tiên tối ưu sẽ kết hợp 2 phương pháp trên cho phép ta đi theo một con đường duy nhất tại một thời điểm, nhưng đồng thời vẫn "quan sát" được những hướng khác. Nếu con đường đang đi "có vẻ" không triển vọng bằng những con đường ta đang "quan sát" ta sẽ chuyển sang đi theo một trong số các con đường này. Để tiện lợi ta sẽ dùng chữ viết tắt BFS thay cho tên gọi tìm kiếm ưu tiên tối ưu.

Một cách cụ thể, tại mỗi bước của tìm kiếm BFS, ta chọn đi theo trạng thái có khả năng cao nhất trong số các trạng thái đã được xét cho đến thời điểm đó. (khác với leo đồi dốc đứng là chỉ chọn trạng thái có khả năng cao nhất trong số các trạng thái kế tiếp có thể đến được từ trạng thái hiện tại). Như vậy, với tiếp cận này, ta sẽ ưu tiên đi vào những nhánh tìm kiếm có khả năng nhất (giống tìm kiếm leo đồi dốc đứng), nhưng ta sẽ không bị lẩn quẩn trong các nhánh này vì nếu càng đi sâu vào một hướng mà ta phát hiện ra rằng hướng này càng đi thì càng tệ, đến mức nó xấu hơn cả những hướng mà ta chưa đi, thì ta sẽ không đi tiếp hướng hiện tại nữa mà chọn đi theo một hướng tốt nhất trong số những hướng chưa đi. Đó là tư tưởng chủ đạo của tìm kiếm BFS. Để hiểu được tư tưởng này. Bạn hãy xem ví dụ sau :



Hình Minh họa thuật giải Best-First Search

Khởi đầu, chỉ có một nút (trạng thái) A nên nó sẽ được mở rộng tạo ra 3 nút mới B,C và D. Các con số dưới nút là giá trị cho biết độ tốt của nút. Con số càng nhỏ, nút càng tốt. Do D là nút có khả năng nhất nên nó sẽ được mở rộng tiếp sau nút A và sinh ra 2 nút kế tiếp là E và F. Đến đây, ta lại thấy nút B có vẻ có khả năng nhất (trong các nút B,C,E,F) nên ta sẽ chọn mở rộng nút B và tạo ra 2 nút G và H. Nhưng lại một lần nữa, hai nút G, H này được đánh giá ít khả năng hơn E, vì thế sự chú ý lại trở về E. E được mở rộng và các nút được sinh ra từ E là I và J. Ở bước kế tiếp, J sẽ được mở rộng vì nó có khả năng nhất. Quá trình này tiếp tục cho đến khi tìm thấy một lời giải.

Lưu ý rằng tìm kiếm này rất giống với tìm kiếm leo đồi dốc đứng, với 2 ngoại lệ. Trong leo núi, một trạng thái được chọn và tất cả các trạng thái khác bị loại bỏ, không bao giờ chúng được xem xét lại. Cách xử lý dứt khoát này là một đặc trưng của leo đồi. Trong BFS, tại một bước, cũng có một di chuyển được chọn nhưng những cái khác vẫn được giữ lại, để ta có thể trở lại xét sau đó khi trạng thái hiện tại trở nên kém khả năng hơn những trạng thái đã được lưu trữ. Hơn nữa, ta chọn trạng thái tốt

nhất mà không quan tâm đến nó có tốt hơn hay không các trạng thái trước đó. Điều này tương phản với leo đồi vì leo đồi sẽ dừng nếu không có trạng thái tiếp theo nào tốt hơn trạng thái hiện hành.

Để cài đặt các thuật giải theo kiểu tìm kiếm BFS, người ta thường cần dùng 2 tập hợp sau :

OPEN : tập chứa các trạng thái đã được sinh ra nhưng chưa được xét đến (vì ta đã chọn một trạng thái khác). Thực ra, OPEN là một loại hàng đợi ưu tiên (priority queue) mà trong đó, phần tử có độ ưu tiên cao nhất là phần tử tốt nhất. Người ta thường cài đặt hàng đợi ưu tiên bằng Heap. Các bạn có thể tham khảo thêm trong các tài liệu về Cấu trúc dữ liệu về loại dữ liệu này.

CLOSE : tập chứa các trạng thái đã được xét đến. Chúng ta cần lưu trữ những trạng thái này trong bộ nhớ để đề phòng trường hợp khi một trạng thái mới được tạo ra lại trùng với một trạng thái mà ta đã xét đến trước đó. Trong trường hợp không gian tìm kiếm có dạng cây thì không cần dùng tập này.



Thuật giải BEST-FIRST SEARCH

1. Đặt OPEN chứa trạng thái khởi đầu.

2. Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong OPEN, thực hiện :

2.a. Chọn trạng thái tốt nhất (Tmax) trong OPEN (và xóa Tmax khỏi OPEN)

2.b. Nếu Tmax là trạng thái kết thúc thì thoát.

2.c. Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện :

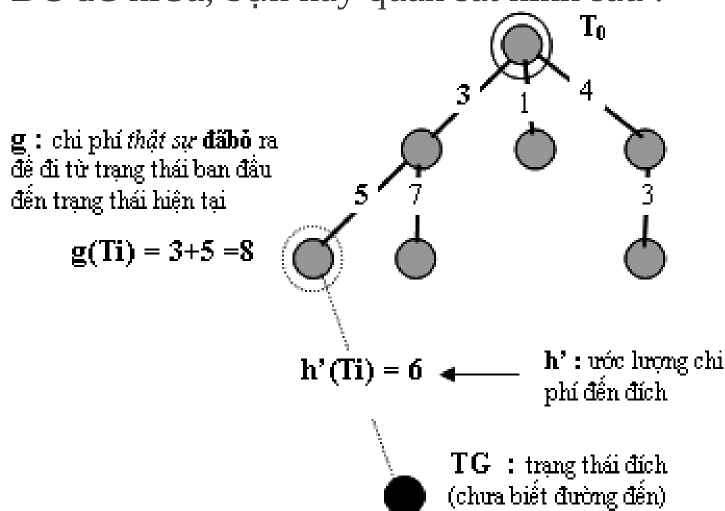
Tính f(Tk); Thêm Tk vào OPEN

BFS khá đơn giản. Tuy vậy, trên thực tế, cũng như tìm kiếm chiều sâu và chiều rộng, hiếm khi ta dùng BFS một cách trực tiếp. Thông thường, người ta thường dùng các phiên bản của BFS là AT, AKT và A*



Thông tin về quá khứ và tương lai

Thông thường, trong các phương án tìm kiếm theo kiểu BFS, độ tốt f của một trạng thái được tính dựa theo 2 hai giá trị mà ta gọi là là g và h'. h' chúng ta đã biết, đó là một ước lượng về chi phí từ trạng thái hiện hành cho đến trạng thái đích (thông tin tương lai). Còn g là "chiều dài quãng đường" đã đi từ trạng thái ban đầu cho đến trạng thái hiện tại (thông tin quá khứ). Lưu ý rằng g là chi phí thực sự (không phải chi phí ước lượng). Để dễ hiểu, bạn hãy quan sát hình sau :



Hình 6.14 Phân biệt khái niệm g và h'

Kết hợp g và h' thành f' (f' = g + h') sẽ thể hiện một ước lượng về "tổng chi phí" cho con đường từ trạng thái bắt đầu đến trạng thái kết thúc dọc theo con đường đi qua trạng thái hiện hành. Để thuận tiện cho thuật giải, ta quy ước là g và h' đều không âm và càng nhỏ nghĩa là càng tốt.

III.5. Thuật giải AT

Thuật giải AT là một phương pháp tìm kiếm theo kiểu BFS với độ tốt của nút là giá trị hàm g – tổng chiều dài con đường đã đi từ trạng thái bắt đầu đến trạng thái hiện tại.



Thuật giải AT

1. Đặt OPEN chứa trạng thái khởi đầu.

2. Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong OPEN, thực hiện :

2.a. Chọn trạng thái (Tmax) có giá trị g nhỏ nhất trong OPEN (và xóa Tmax khỏi OPEN)

2.b. Nếu Tmax là trạng thái kết thúc thì thoát.

2.c. Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện :

g(Tk) = g(Tmax) + cost(Tmax, Tk);

Thêm Tk vào OPEN.

* Vì chỉ sử dụng hàm g (mà không dùng hàm ước lượng h') fsđể đánh giá độ tốt của một trạng thái nên ta cũng có thể xem AT chỉ là một thuật toán.

III.6. Thuật giải AKT

(Algorithm for Knowlegeable Tree Search)

Thuật giải AKT mở rộng AT bằng cách sử dụng thêm thông tin ước lượng h'. Độ tốt của một trạng thái f là tổng của hai hàm g và h'.



Thuật giải AKT

1. Đặt OPEN chứa trạng thái khởi đầu.

2. Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong OPEN, thực hiện :

2.a. Chọn trạng thái (Tmax) có giá trị f nhỏ nhất trong OPEN (và xóa Tmax khỏi OPEN)

2.b. Nếu Tmax là trạng thái kết thúc thì thoát.

2.c. Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái Tmax. Đối với mỗi trạng thái kế tiếp Tk thực hiện :

g(Tk) = g(Tmax) + cost(Tmax, Tk);

Tính h’(Tk)

f(Tk) = g(Tk) + h’(Tk);

Thêm Tk vào OPEN.

III.7. Thuật giải A*

A* là một phiên bản đặc biệt của AKT áp dụng cho trường hợp đồ thị. Thuật giải A* có sử dụng thêm tập hợp CLOSE để lưu trữ những trường hợp đã được xét đến. A* mở rộng AKT bằng cách bổ sung cách giải quyết trường hợp khi "mở" một nút mà nút này đã có sẵn trong OPEN hoặc CLOSE. Khi xét đến một trạng thái Ti bên cạnh việc lưu trữ 3 giá trị cơ bản g,h’, f’ để phản ánh độ tốt của trạng thái đó, A* còn lưu trữ thêm hai thông số sau :

1. Trạng thái cha của trạng thái Ti (ký hiệu là Cha(Ti) : cho biết trạng thái dẫn đến trạng thái Ti. Trong trường hợp có nhiều trạng thái dẫn đến Ti thì chọn Cha(Ti) sao cho chi phí đi từ trạng thái khởi đầu đến Ti là thấp nhất, nghĩa là :

g(Ti) = g(Tcha) + cost(Tcha, Ti) là thấp nhất.

2. Danh sách các trạng thái kế tiếp của Ti : danh sách này lưu trữ các trạng thái kế tiếp Tk của Ti sao cho chi phí đến Tk thông qua Ti từ trạng thái ban đầu là thấp nhất. Thực chất thì danh sách này có thể được tính ra từ thuộc tính Cha của các trạng thái được lưu trữ. Tuy nhiên, việc tính toán này có thể mất nhiều thời gian (khi tập OPEN, CLOSE được mở rộng) nên người ta thường lưu trữ ra một danh sách riêng. Trong thuật toán sau đây, chúng ta sẽ không đề cập đến việc lưu trữ danh sách này. Sau khi hiểu rõ thuật toán, bạn đọc có thể dễ dàng điều chỉnh lại thuật toán để lưu trữ thêm thuộc tính này.

1. Đặt OPEN chỉ chứa T0. Đặt g(T0) = 0, h'(T0) = 0 và f'(T0) = 0. Đặt CLOSE là tập hợp rỗng.

2. Lặp lại các bước sau cho đến khi gặp điều kiện dừng.

2.a. Nếu OPEN rỗng : bài toán vô nghiệm, thoát.

2.b. Ngược lại, chọn Tmax trong OPEN sao cho f'(Tmax) là nhỏ nhất

2.b.1. Lấy Tmax ra khỏi OPEN và đưa Tmax vào CLOSE.

2.b.2. Nếu Tmax chính là TG thì thoát và thông báo lời giải là Tmax.

2.b.3. Nếu Tmax không phải là TG. Tạo ra danh sách tất cả các trạng thái kế tiếp của Tmax. Gọi một trạng thái này là Tk. Với mỗi Tk, làm các bước sau :

2.b.3.1. Tính g(Tk) = g(Tmax) + cost(Tmax, Tk).

2.b.3.2. Nếu tồn tại Tk' trong OPEN trùng với Tk.

Nếu g(Tk) < g(Tk') thì

Đặt g(Tk') = g(Tk)

Tính lại f'(Tk')

Đặt Cha(Tk’) = Tmax

2.b.3.3. Nếu tồn tại Tk’ trong CLOSE trùng với Tk.

Nếu g(Tk) < g(Tk’) thì

Đặt g(Tk’) = g(Tk)

Tính lại f’(Tk’)

Đặt Cha(Tk’) = Tmax

Lan truyền sự thay đổi giá trị g, f’ cho tất cả các trạng thái kế tiếp của Ti (ở tất cả các cấp) đã được lưu trữ trong CLOSE và OPEN.

2.b.3.4. Nếu Tk chưa xuất hiện trong cả OPEN lẫn CLOSE thì :

Thêm Tk vào OPEN

Tính : f' (Tk) = g(Tk)+h’(Tk).

Có một số điểm cần giải thích trong thuật giải này. Đầu tiên là việc sau khi đã tìm thấy trạng thái đích TG, làm sao để xây dựng lại được "con đường" từ T0 đến TG. Rất đơn giản, bạn chỉ cần lần ngược theo thuộc tính Cha của các trạng thái đã được lưu trữ trong CLOSE cho đến khi đạt đến T0. Đó chính là "con đường" tối ưu đi từ TG đến T0 (hay nói cách khác là từ T0 đến TG).

Điểm thứ hai là thao tác cập nhật lại g(Tk’) , f’(Tk’) và Cha(Tk’) trong bước 2.b.3.2 và 2.b.3.3. Các thao tác này thể hiện tư tưởng : "luôn chọn con đường tối ưu nhất". Như chúng ta đã biết, giá trị g(Tk’) nhằm lưu trữ chi phí tối ưu thực sự tính từ T0 đến Tk’. Do đó, nếu chúng ta phát hiện thấy một "con đường" khác tốt hơn thông qua Tk (có chi phí nhỏ hơn) con đường hiện tại được lưu trữ thì ta phải chọn "con đường" mới tốt hơn này. Trường hợp 2.b.3.3 phức tạp hơn. Vì từ Tk’ nằm trong tập CLOSE nên từ Tk’ ta đã lưu trữ các trạng thái con kế tiếp xuất phát từ Tk’. Nhưng g(Tk’) thay đổi dẫn đến giá trị g của các trạng thái con này

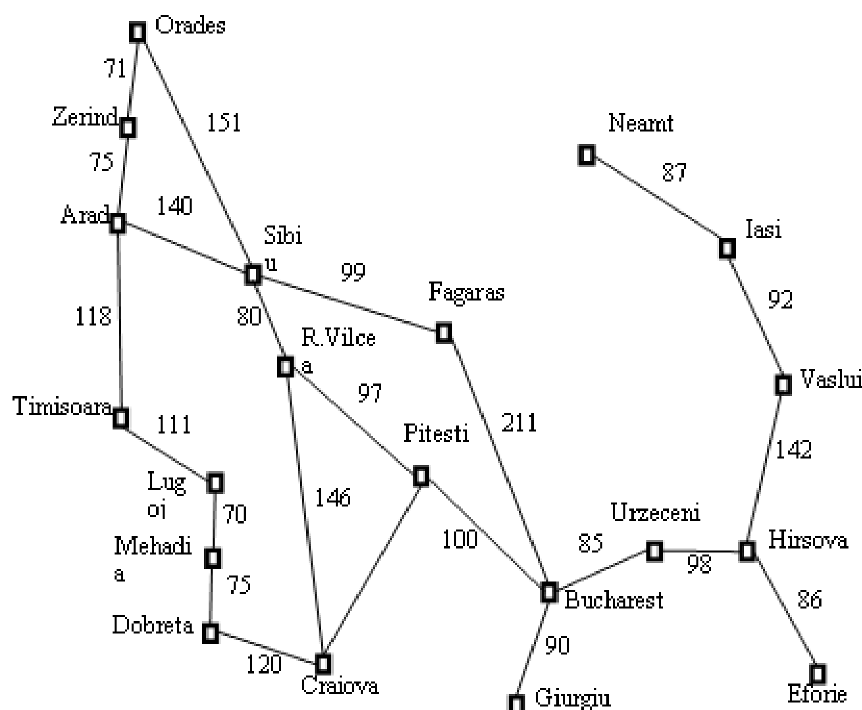
cũng phải thay đổi theo. Và đến lượt các trạng thái con này lại có thể có các các trạng thái con tiếp theo của chúng và cứ thế cho đến khi mỗi nhánh kết thúc với một trạng thái trong OPEN (nghĩa là không có trạng thái con nào nữa). Để thực hiện quá trình cập nhật này, ta hãy thực hiện quá trình duyệt theo chiều sâu với điểm khởi đầu là Tk’. Duyệt đến đâu, ta cập nhật lại g của các trạng thái đến đó (dùng công thức g(T) = g(Cha(T)) +cost(Cha(T), T)) và vì thế giá trị f’ của các trạng thái này cũng thay đổi theo.

Một lần nữa, xin nhắc lại rằng, bạn có thể cho rằng tập OPEN lưu trữ các trạng thái "sẽ được xem xét đến sau" còn tập CLOSE lưu trữ các trạng thái "đã được xét đến rồi".

Có thể bạn sẽ cảm thấy khá lúng túng trước một thuật giải dài như thế. Vấn đề có lẽ sẻ trở nên sáng sủa hơn khi bạn quan sát các bước giải bài toán tìm đường đi ngắn nhất trên đồ thị bằng thuật giải A* sau đây.

III.8. Ví dụ minh họa hoạt động của thuật giải A*

Chúng ta sẽ minh họa hoạt động của thuật giải A* trong việc tìm kiếm đường đi ngắn nhất từ thành phố Arad đến thành phố Bucharest của Romania. Bản đồ các thành phố của Romania được cho trong đồ thị sau. Trong đó mỗi đỉnh của đồ thị của là một thành phố, giữa hai đỉnh có cung nối nghĩa là có đường đi giữa hai thành phố tương ứng. Trọng số của cung chính là chiều dài (tính bằng km) của đường đi nối hai thành phố tương ứng, chiều dài theo đường chim bay một thành phố đến Bucharest được cho trong bảng kèm theo.



Hình : Bảng đồ của Romania với khoảng cách đường tính theo km

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	98
Eforie	161	R.Vilcea	193
Fagaras	178	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Bảng : Khoảng cách đường chim bay từ một thành phố đến Bucharest.

Chúng ta sẽ chọn hàm h’ chính là khoảng cách đường chim bay cho trong bảng trên và hàm chi phí cost(Ti, Ti+1) chính là chiều dài con đường nối từ thành phố Ti và Ti+1.

Sau đây là từng bước hoạt động của thuật toán A* trong việc tìm đường đi ngắn nhất từ Arad đến Bucharest.

Ban đầu :

OPEN {(Arad,g 0,h’ 0,f’ 0)}

CLOSE {}

Do trong OPEN chỉ chứa một thành phố duy nhất nên thành phố này sẽ là thành phố tốt nhất. Nghĩa là Tmax Arad.Ta lấy Arad ra khỏi OPEN và đưa vào CLOSE.

OPEN {}

CLOSE {(Arad,g 0,h’ 0,f’ 0)}

Từ Arad có thể đi đến được 3 thành phố là Sibiu, Timisoara và Zerind. Ta lần lượt tính giá trị f’, g và h’ của 3 thành phố này. Do cả 3 nút mới tạo ra này chưa có nút cha nên ban đầu nút cha của chúng đều là Arad.

h’(Sibiu) 253

g(Sibiu) g(Arad)+cost(Arad,Sibiu)

0+140 140

f’(Sibiu) g(Sibiu)+h’(Sibiu)

140+253 393

Cha(Sibiu) Arad

h’(Timisoara) 329

g(Timisoara) g(Arad)+cost(Arad, Timisoara)

0+118 118

f’(Timisoara) g(Timisoara)+ h’(Timisoara)

118+329 447

Cha(Timisoara) Arad

h’(Zerind)   374

g(Zerind)   g(Arad)+cost(Arad, Zerind)

  0+75 75

f’(Zerind)   g(Zerind)+h’(Zerind)

  75+374   449

Cha(Zerind)   Arad

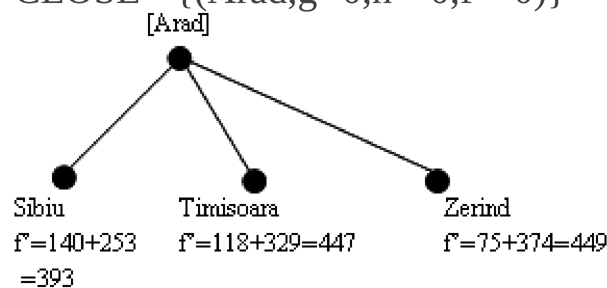
Do cả 3 nút Sibiu, Timisoara, Zerind đều không có trong cả OPEN và CLOSE nên ta bổ sung 3 nút này vào OPEN.

OPEN   {(Sibiu,g 140,h’ 253,f’ 393,Cha Arad)

(Timisoara,g 118,h’ 329,f’ 447,Cha Arad)

(Zerind,g 75,h’ 374,f’ 449,Cha Arad)}

CLOSE   {(Arad,g 0,h’ 0,f’ 0)}



Hình : Bước 1, nút được đóng ngoặc vuông (như [Arad]) là nút trong tập CLOSE, ngược lại là trong tập OPEN.

Trong tập OPEN, nút Sibiu là nút có giá trị f’ nhỏ nhất nên ta sẽ chọn Tmax   Sibiu. Ta lấy Sibiu ra khỏi OPEN và đưa vào CLOSE.

OPEN   {(Timisoara,g 118,h’ 329,f’ 447,Cha Arad)

(Zerind,g 75,h’ 374,f’ 449,Cha Arad)}

CLOSE {(Arad,g 0,h' 0,f' 0)

(Sibiu,g 140,h' 253,f' 393,Cha Arad)}

Từ Sibiu có thể đi đến được 4 thành phố là : Arad, Fagaras, Oradea, Rimnicu. Ta lần lượt tính các giá trị g, h', f' cho các nút này.

h'(Arad) 366

g(Arad) g(Sibiu)+cost(Sibiu,Arad)

 140+140 280

f'(Arad) g(Arad)+h'(Arad)

 280+366 646

h'(Fagaras) 178

g(Fagaras) g(Sibiu)+cost(Sibiu, Fagaras) 140+99 239

f'(Fagaras) g(Fagaras)+ h'(Fagaras)

 239+178 417

h'(Oradea) 380

g(Oradea) g(Sibiu)+cost(Sibiu, Oradea)

 140+151 291

f'(Oradea) g(Oradea)+ h'(Oradea)

 291+380 671

h'(R.Vilcea) 193

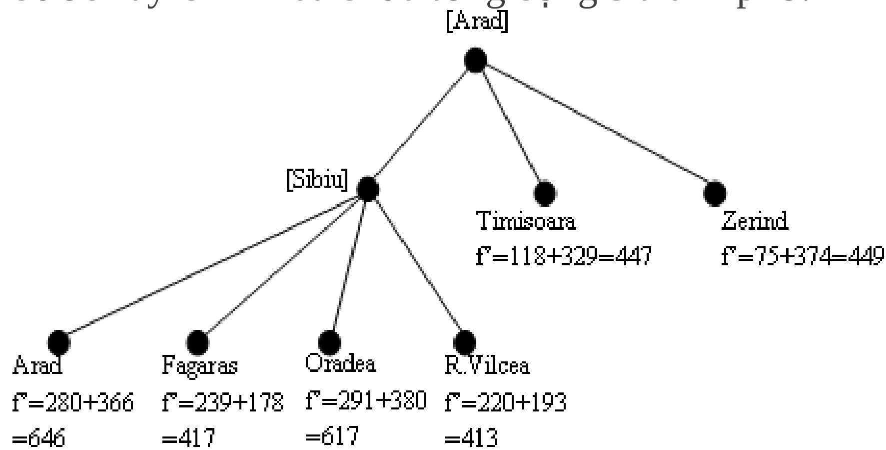
g(R.Vilcea) g(Sibiu)+cost(Sibiu, R.Vilcea)

140+80   220

f’(R.Vilcea)   g(R.Vilcea)+ h’(R.Vilcea)

220+193   413

Nút Arad đã có trong CLOSE. Tuy nhiên, do g(Arad) mới được tạo ra (có giá trị 280) lớn hơn g(Arad) lưu trong CLOSE (có giá trị 0) nên ta sẽ không cập nhật lại giá trị g và f’ của Arad lưu trong CLOSE. 3 nút còn lại : Fagaras, Oradea, Rimnicu đều không có trong cả OPEN và CLOSE nên ta sẽ đưa 3 nút này vào OPEN, đặt cha của chúng là Sibiu. Như vậy, đến bước này OPEN đã chứa tổng cộng 5 thành phố.



OPEN   {(Timisoara,g   118,h’   329,f’   447,Cha   Arad)

(Zerind,g   75,h’   374,f’   449,Cha   Arad)

(Fagaras,g   239,h’   178,f’   417,Cha   Sibiu)

(Oradea,g   291,h’   380,f’   617,Cha   Sibiu)

(R.Vilcea,g   220,h’   193,f’   413,Cha   Sibiu)}

CLOSE   {(Arad,g   0,h’   0,f’   0)

(Sibiu,g   140,h’   253,f’   393,Cha   Arad)}

Trong tập OPEN, nút R.Vilcea là nút có giá trị f’ nhỏ nhất. Ta chọn Tmax R.Vilcea. Chuyển R.Vilcea từ OPEN sang CLOSE. Từ R.Vilcea có thể đi đến được 3 thành phố là Craiova, Pitesti và Sibiu. Ta lần lượt tính giá trị f’, g và h’ của 3 thành phố này.

h’(Sibiu) 253

g(Sibiu) g(R.Vilcea)+ cost(R.Vilcea,Sibiu)

220+80 300

f’(Sibiu) g(Sibiu)+h’(Sibiu)

300+253 553

h’(Craiova) 160

g(Craiova) g(R.Vilcea)+ cost(R.Vilcea, Craiova)

220+146 366

f’(Craiova) g(Fagaras)+h’(Fagaras)

366+160 526

h’(Pitesti) 98

g(Pitesti) g(R.Vilcea)+ cost(R.Vilcea, Pitesti)

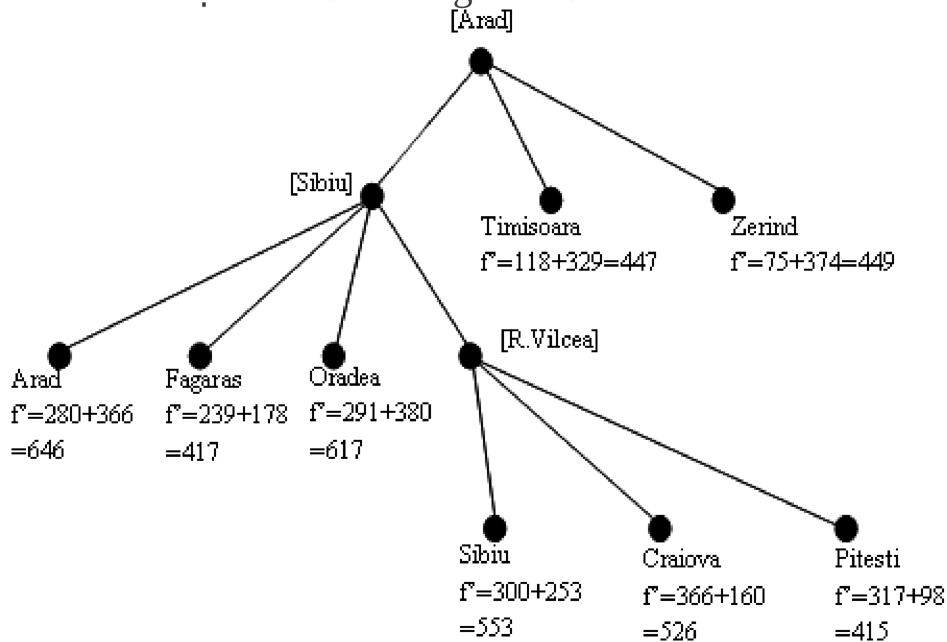
220+97 317

f’(Pitesti) g(Oradea)+h’(Oradea)

317+98 415

Sibiu đã có trong tập CLOSE. Tuy nhiên, do g’(Sibiu) mới (có giá trị là 553) lớn hơn g’(Sibiu) (có giá trị là 393) nên ta sẽ không cập nhật lại các giá trị của Sibiu được lưu trong CLOSE. Còn lại 2 thành phố là Pitesti và

Craiova đều không có trong cả OPEN và CLOSE nên ta sẽ đưa nó vào OPEN và đặt cha của chúng là R.Vilcea.



OPEN {(Timisoara,g 118,h' 329,f' 447,Cha Arad)

(Zerind,g 75,h' 374,f' 449,Cha Arad) (Fagaras,g 239,h' 178,f' 417,Cha Sibiu)

(Oradea,g 291,h' 380,f' 617,Cha Sibiu) (Craiova,g 366,h' 160,f' 526,Cha R.Vilcea)

(Pitesti,g 317,h' 98,f' 415,Cha R.Vilcea) }

CLOSE {(Arad,g 0,h' 0,f' 0)

(Sibiu,g 140,h' 253,f' 393,Cha Arad)

(R.Vilcea,g 220,h' 193,f' 413,Cha Sibiu) }

Đến đây, trong tập OPEN, nút tốt nhất là Pitesti, từ Pitesti ta có thể đi đến được R.Vilcea, Bucharest và Craiova. Lấy Pitesti ra khỏi OPEN và đặt nó vào CLOSE. Thực hiện tiếp theo tương tự như trên, ta sẽ không cập nhật giá trị f', g của R.Vilcea và Craiova lưu trong CLOSE. Sau khi tính toán f',

g của Bucharest, ta sẽ đưa Bucharest vào tập OPEN, đặt Cha(Bucharest) Pitesti.

h’(Bucharest)   0

g(Bucharest)   g(Pitesti)+cost(Pitesti, Bucharest)

  317+100 418

f’(Bucharest)   g(Fagaras)+h’(Fagaras)

  417+0 417

Ở bước kế tiếp, ta sẽ chọn được Tmax   Bucharest. Và như vậy thuật toán kết thúc (thực ra thì tại bước này, có hai ứng cử viên là Bucharest và Fagaras vì đều cùng có f’ 417 , nhưng vì Bucharest là đích nên ta sẽ ưu tiên chọn hơn).

Để xây dựng lại con đường đi từ Arad đến Bucharest ta lần theo giá trị Cha được lưu trữ kèm với f’, g và h’ cho đến lúc đến Arad.

Cha(Bucharest)   Pitesti

Cha(R.Vilcea)   Sibiu

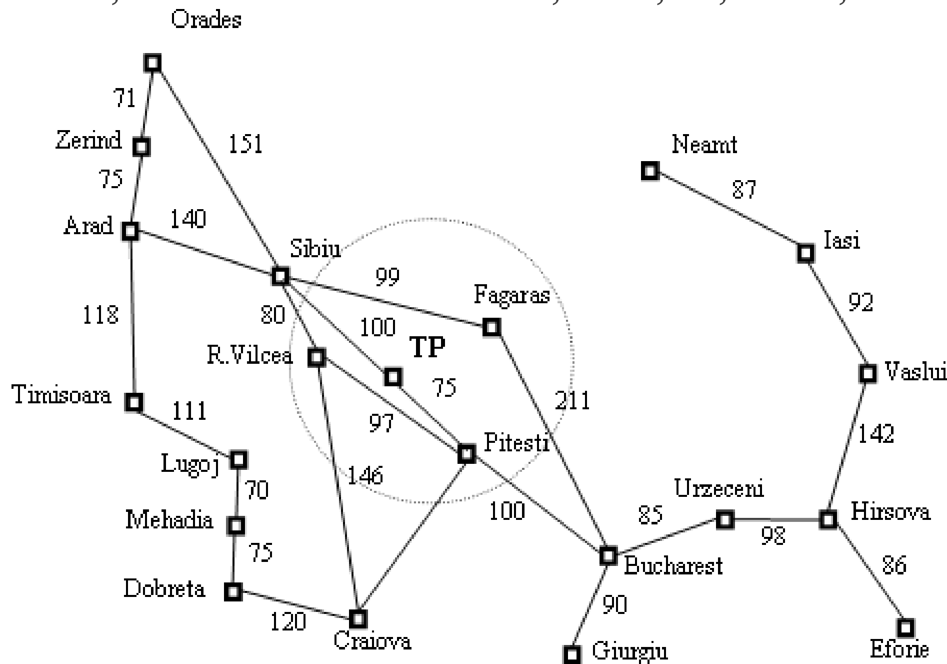
Cha(Sibiu)   Arad

Vậy con đường đi ngắn nhất từ Arad đến Bucharest là Arad, Sibiu, R.Vilcea, Pitesti, Bucharest.

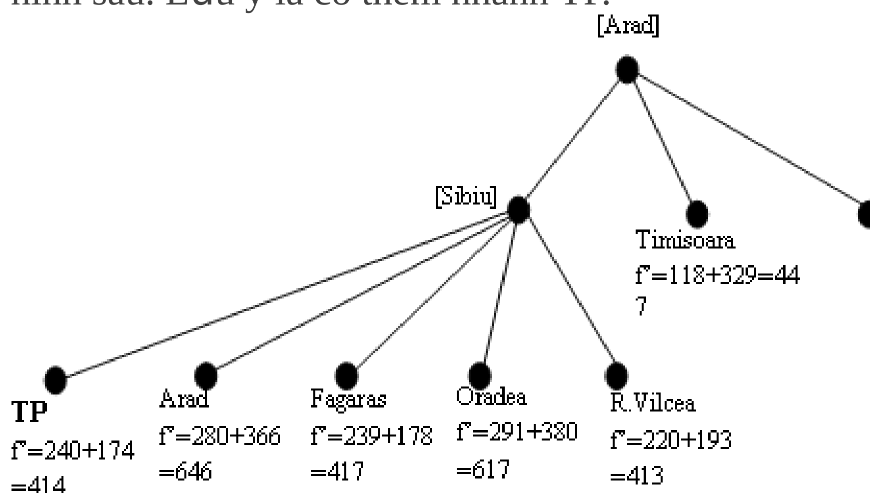
Trong ví dụ minh họa này, hàm h’ có chất lượng khá tốt và cấu trúc đồ thị khá đơn giản nên ta gần như đi thẳng đến đích mà ít phải khảo sát các con đường khác. Đây là một trường hợp đơn giản, trong trường hợp này, thuật giải có dáng dấp của tìm kiếm chiều sâu.

Đến đây, để minh họa một trường hợp phức tạp hơn của thuật giải. Ta thử sửa đổi lại cấu trúc đồ thị và quan sát hoạt động của thuật giải. Giả sử ta có thêm một thành phố tạm gọi là TP và con đường giữa Sibiu và

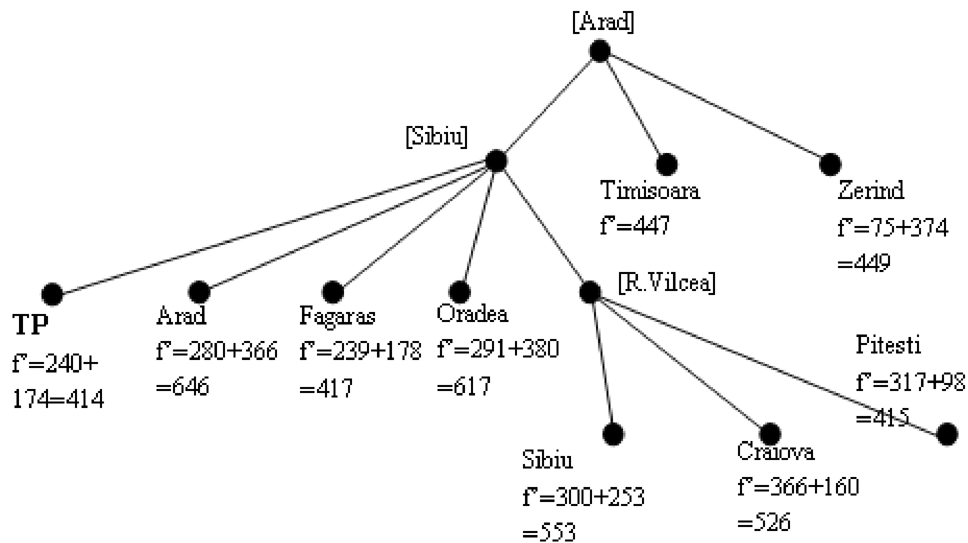
TP có chiều dài 100, con đường giữa TP và Pitesti có chiều dài 60. Và khoảng cách đường chim bay từ TP đến Bucharest là 174. Như vậy rõ ràng, con đường tối ưu đến Bucharest không còn là Arad, Sibiu, R.Vilcea, Pitesti, Bucharest nữa mà là Arad, Sibiu, TP, Pitesti, Bucharest.



Trong trường hợp này, chúng ta vẫn tiến hành bước 1 như ở trên. Sau khi thực hiện hiện bước 2 (mở rộng Sibiu), chúng ta có cây tìm kiếm như hình sau. Lưu ý là có thêm nhánh TP.



R.Vilcea vẫn có giá trị f' thấp nhất. Nên ta mở rộng R.Vilcea như trường hợp đầu tiên.



Bước kế tiếp của trường hợp đơn giản là mở rộng Pitesti để có được kết quả. Tuy nhiên, trong trường hợp này, TP có giá trị f’ thấp hơn. Do đó, ta chọn mở rộng TP. Từ TP ta chỉ có 2 hướng đi, một quay lại Sibiu và một đến Pitesti. Để nhanh chóng, ta sẽ không tính toán giá trị của Sibiu vì biết chắc nó sẽ lớn hơn giá trị được lưu trữ trong CLOSE (vì đi ngược lại).

h’(Pitesti)   98

g(Pitesti)   g(TP)+cost(TP, Pitesti)

  240+75 315

f’(Pitesti)   g(TP)+h’(Pitesti)   315+98 413

Pistestti đã xuất hiện trong tập OPEN và g’(Pitesti) mới (có giá trị là 315) thấp hơn g’(Pitesti) cũ (có giá trị 317) nên ta phải cập nhật lại giá trị của f’,g, Cha của Pitesti lưu trong OPEN. Sau khi cập nhật xong, tập OPEN và CLOSE sẽ như sau :

OPEN   {(Timisoara,g 118,h’ 329,f’ 447,Cha Arad)

(Zerind,g 75,h’ 374,f’ 449,Cha Arad)

(Fagaras,g 239,h’ 178,f’ 417,Cha Sibiu)

(Oradea,g 291,h’ 380,f’ 617,Cha Sibiu)

(Craiova,g 366,h’ 160,f’ 526,Cha R.Vilcea)

(Pitesti,g 315,h’ 98,f’ 413,Cha TP) }

CLOSE {(Arad,g 0,h’ 0,f’ 0)

(Sibiu,g 140,h’ 253,f’ 393,Cha Arad)

(R.Vilcea,g 220,h’ 193,f’ 413,Cha Sibiu)

}

Đến đây ta thấy rằng, ban đầu thuật giải chọn đường đi đến Pitesti qua R.Vilcea. Tuy nhiên, sau đó, thuật giải phát hiện ra con đường đến Pitesti qua TP là tốt hơn nên nó sẽ sử dụng con đường này. Đây chính là trường hợp 2.b.iii.2 trong thuật giải.

Bước sau, chúng ta sẽ chọn mở rộng Pitesti như bình thường. Khi lần ngược theo thuộc tính Cha, ta sẽ có con đường tối ưu là Arad, Sibiu, TP, Pitesti, Bucharest.

III.9. Bàn luận về A*

Đến đây, có lẽ bạn đã hiểu được thuật giải này. Ta có một vài nhận xét khá thú vị về A*. Đầu tiên là vai trò của g trong việc giúp chúng ta lựa chọn đường đi. Nó cho chúng ta khả năng lựa chọn trạng thái nào để mở rộng tiếp theo, không chỉ dựa trên việc trạng thái đó tốt như thế nào (thể hiện bởi giá trị h’) mà còn trên cơ sở con đường từ trạng thái khởi đầu đến trạng thái hiện tại đó tốt ra sao. Điều này sẽ rất hữu ích nếu ta không chỉ quan tâm việc tìm ra lời giải hay không mà còn quan tâm đến hiệu quả của con đường dẫn đến lời giải. Chẳng hạn như trong bài toán tìm đường đi ngắn nhất giữa hai điểm. Bên cạnh việc tìm ra đường đi giữa hai điểm, ta còn phải tìm ra một con đường ngắn nhất. Tuy nhiên, nếu ta chỉ quan tâm đến việc tìm được lời giải (mà không quan tâm đến hiệu quả của con đường đến lời giải), chúng ta có thể đặt g=0 ở mọi trạng thái. Điều này sẽ giúp ta luôn chọn đi theo trạng thái có vẻ gần

nhất với trạng thái kết thúc (vì lúc này f' chỉ phụ thuộc vào h' là hàm ước lượng "khoảng cách" gần nhất để tới đích). Lúc này thuật giải có dáng dấp của tìm kiếm chiều sâu theo nguyên lý hướng đích kết hợp với lần ngược.

Ngược lại, nếu ta muốn tìm ra kết quả với số bước ít nhất (đạt được trạng thái đích với số trạng thái trung gian ít nhất), thì ta đặt giá trị để đi từ một trạng thái đến các trạng thái con kế tiếp của nó luôn là hằng số, thường là 1 Nghĩa đặt cost(Ti-1, Ti) = 1 (và vẫn dùng một hàm ước lượng h' như bình thường). Còn ngược lại, nếu muốn tìm chi phí rẻ nhất thì ta phải đặt giá trị hàm cost chính xác (phản ánh đúng ghi phí thực sự).

Đến đây, chắc bạn đọc đã có thể bắt đầu cảm nhận được rằng thuật giải A* không hoàn toàn là một thuật giải tối ưu tuyệt đối. Nói đúng hơn, A* chỉ là một thuật giải linh động và cho chúng ta khá nhiều tùy chọn. Tùy theo bài toán mà ta sẽ có một bộ thông số thích hợp cho A* để thuật giải hoạt động hiệu quả nhất.

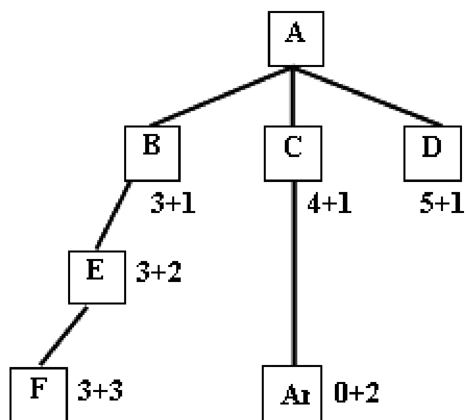
Điểm quan tâm thứ hai là về giá trị h' – sự ước lượng khoảng cách (chi phí) từ một trạng thái đến trạng thái đích. Nếu h' chính là h (đánh giá tuyệt đối chính xác) thì A* sẽ đi một mạch từ trạng thái đầu đến trạng thái kết thúc mà không cần phải thực hiện bất kỳ một thao tác đổi hướng nào!. Dĩ nhiên, trên thực tế, hầu như chẳng bao giờ ta tìm thấy một đánh giá tuyệt đối chính xác. Tuy nhiên, điều đáng quan tâm ở đây là h' được ước lượng càng gần với h, quá trình tìm kiếm càng ít bị sai sót, ít bị rẽ vào những nhánh cụt hơn. Hay nói ngắn gọn là càng nhanh chóng tìm thấy lời giải hơn.

Nếu h' luôn bằng 0 ở mọi trạng thái (trở về thuật giải AT) thì quá trình tìm kiếm sẽ được điều khiển hoàn toàn bởi giá trị g. Nghĩa là thuật giải sẽ chọn đi theo những hướng mà sẽ tốn ít chi phí/bước đi nhất (chi phí tính từ trạng thái đầu tiên đến trạng thái hiện đang xét) bất chấp việc đi theo hướng đó có khả năng dẫn đến lời giải hay không. Đây chính là hình ảnh của nguyên lý tham lam (Greedy).

Nếu chi phí từ trạng thái sang trạng thái khác luôn là hằng số (dĩ nhiên lúc này h’ luôn bằng 0) thì thuật giải A* trở thành thuật giải tìm kiếm theo chiều rộng! Lý do là vì tất cả những trạng thái cách trạng thái khởi đầu n bước đều có cùng giá trị g và vì thế đều có cùng f’ và giá trị này sẽ nhỏ hơn tất cả các trạng thái cách trạng thái khởi đầu n+1 bước. Và nếu g luôn bằng 0 và h’ cũng luôn bằng 0, mọi trạng thái đang xét đều tương đương nhau. Ta chỉ có thể chọn bằng trạng thái kế tiếp bằng ngẫu nhiên !

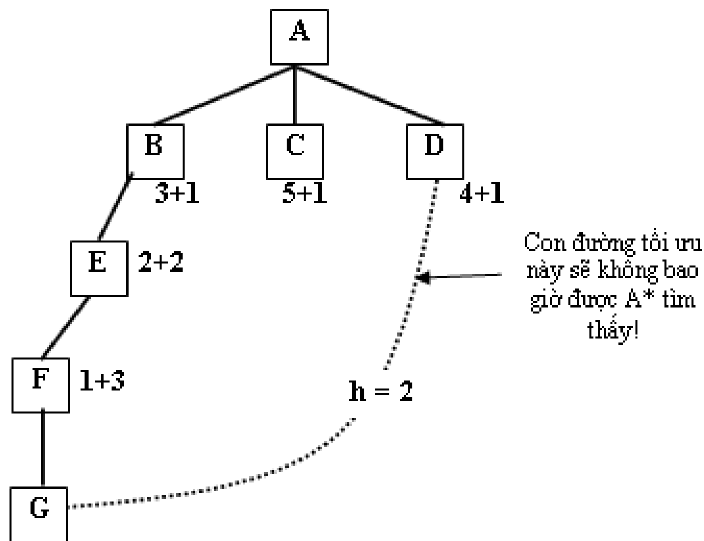
Còn nếu như h’ không thể tuyệt đối chính xác (nghĩa là không bằng đúng h) và cũng không luôn bằng 0 thì sao? Có điều gì thú vị về cách xử lý của quá trình tìm kiếm hay không? Câu trả lời là có. Nếu như bằng một cách nào đó, ta có thể chắc chắn rằng, ước lượng h’ luôn nhỏ hơn h (đối với mọi trạng thái) thì thuật giải A* sẽ thường tìm ra con đường tối ưu (xác định bởi g) để đi đến đích, nếu đường dẫn đó tồn tại và quá trình tìm kiếm sẽ ít khi bị sa lầy vào những con đường quá dở. Còn nếu vì một lý do nào đó, ước lượng h’ lại lớn hơn h thì thuật giải sẽ dễ dàng bị vướng vào những hướng tìm kiếm vô ích. Thậm chí nó lại có khuynh hướng tìm kiếm ở những hướng đi vô ích trước! Điều này có thể thấy một cách dễ dàng từ vài ví dụ.

Xét trường hợp được trình bày trong hình sau. Giả sử rằng tất cả các cung đều có giá trị 1. G là trạng thái đích. Khởi đầu, OPEN chỉ chứa A, sau đó A được mở rộng nên B, C, D sẽ được đưa vào OPEN (hình vẽ mô tả trạng thái 2 bước sau đó, khi B và E đã được mở rộng). Đối với mỗi nút, con số đầu tiên là giá trị h’, con số kế tiếp là g. Trong ví dụ này, nút B có f’ thấp nhất là 4 = h’+g = 3 + 1 , vì thế nó được mở rộng trước tiên. Giả sử nó chỉ có một nút con tiếp theo là E và h’(E) = 3, do E các A hai cung nên g(E) = 2 suy ra f’(E) = 5, giống như f’(C). Ta chọn mở rộng E kế tiếp. Giả sử nó cũng chỉ có duy nhất một con kế tiếp là F và h’(F) cũng bằng 3. Rõ ràng là chúng ta đang di chuyển xuống và không phát triển rộng. Nhưng f’(F) = 6 lớn hơn f’(D). Do đó, chúng ta sẽ mở rộng C tiếp theo và đạt đến trạng thái đích. Như vậy, ta thấy rằng do đánh giá thấp h(B) nên ta đã lãng phí một số bước (E,F), nhưng cuối cùng ta cùng phát hiện ra B khác xa với điều ta mong đợi và quay lại để thử một đường dẫn khác.



Hình : h’ đánh giá thấp h

Bây giờ hãy xét trường hợp ở hình tiếp theo. Chúng ta cũng mở rộng B ở bước đầu tiên và E ở bước thứ hai. Kế tiếp là F và cuối cùng G, cho đường dẫn kết thúc có độ dài là 4. Nhưng giả sử có đường dẫn trực tiếp từ D đến một lời giải có độ dài h thực sự là 2 thì chúng ta sẽ không bao giờ tìm được đường dẫn này (tuy rằng ta có thể tìm thấy lời giải). Bởi vì việc đánh giá quá cao h’(D), chúng ta sẽ làm cho D trông dở đến nỗi mà ta phải tìm một đường đi khác – đến một lời giải tệ hơn - mà không bao giờ nghĩ đến việc mở rộng D. Nói chung, nếu h’ đánh giá cao h thì A* sẽ có thể không thể tìm ra đường dẫn tối ưu đến lời giải (nếu như có nhiều đường dẫn đến lời giải). Một câu hỏi thú vị là "Liệu có một nguyên tắc chung nào giúp chúng ta đưa ra một cách ước lượng h’ không bao giờ đánh giá cao h hay không?". Câu trả lời là "hầu như không", bởi vì đối với hầu hết các vấn đề thực ta đều không biết h. Tuy nhiên, cách duy nhất để bảo đảm h’ không bao giờ đánh giá cao h là đặt h’ bằng 0 !



Hình : h’ đánh giá cao h

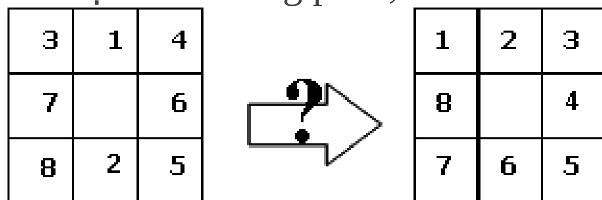
Đến đây chúng ta đã kết thúc việc bàn luận về thuật giải A*, một thuật giải linh động, tổng quát, trong đó hàm chứa cả tìm kiếm chiều sâu, tìm kiếm chiều rộng và những nguyên lý Heuristic khác. Chính vì thế mà người ta thường nói, A* chính là thuật giải tiêu biểu cho Heuristic.

A* rất linh động nhưng vẫn gặp một khuyết điểm cơ bản – giống như chiến lược tìm kiếm chiều rộng – đó là tốn khá nhiều bộ nhớ để lưu lại những trạng thái đã đi qua – nếu chúng ta muốn nó chắc chắn tìm thấy lời giải tối ưu. Với những không gian tìm kiếm lớn nhỏ thì đây không phải là một điểm đáng quan tâm. Tuy nhiên, với những không gian tìm kiếm khổng lồ (chẳng hạn tìm đường đi trên một ma trận kích thước cỡ 106 x 106) thì không gian lưu trữ là cả một vấn đề hóc búa. Các nhà nghiên cứu đã đưa ra khá nhiều các hướng tiếp cận lai để giải quyết vấn đề này. Chúng ta sẽ tìm hiểu một số phương án nhưng quan trọng nhất, ta cần phải nắm rõ vị trí của A* so với những thuật giải khác.

III.10. Ứng dụng A* để giải bài toán Ta-canh

Bài toán Ta-canh đã từng là một trò chơi khá phổ biến, đôi lúc người ta còn gọi đây là bài toán 9-puzzle. Trò chơi bao gồm một hình vuông kích thước 3x3 ô. Có 8 ô có số, mỗi ô có một số từ 1 đến 8. Một ô còn trống. Mỗi lần di chuyển chỉ được di chuyển một ô nằm cạnh ô trống về phía ô

trống. Vấn đề là từ một trạng thái ban đầu bất kỳ, làm sao đưa được về trạng thái cuối là trạng thái mà các ô được sắp lần lượt từ 1 đến 8 theo thứ tự từ trái sang phải, từ trên xuống dưới, ô cuối dùng là ô trống.



Cho đến nay, ngoại trừ 2 giải pháp vét cạn và tìm kiếm Heuristic, người ta vẫn chưa tìm được một thuật toán chính xác, tối ưu để giải bài toán này. Tuy nhiên, cách giải theo thuật giải A* lại khá đơn giản và thường tìm được lời giải (nhưng không phải lúc nào cũng tìm được lời giải). Nhận xét rằng: Tại mỗi thời điểm ta chỉ có tối đa 4 ô có thể di chuyển. Vấn đề là tại thời điểm đó, ta sẽ chọn lựa di chuyển ô nào? Chẳng hạn ở hình trên, ta nên di chuyển (1), (2), (6), hay (7) ? Bài toán này hoàn toàn có cấu trúc thích hợp để có thể giải bằng A* (tổng số trạng thái có thể có của bàn cờ là n2! với n là kích thước bàn cờ vì mỗi trạng thái là một hoán vị của tập n2 con số).

Tại một trạng thái đang xét Tk, đặt d(i,j)là số ô cần di chuyển để đưa con số ở ô (i,j) về đúng vị trí của nó ở trạng thái đích.

Hàm ước lượng h’ tại trạng thái Tk bất kỳ bằng tổng của các d(i,j) sao cho vị trí (i,j) không phải là ô trống.

Như vậy đối với trạng thái ở hình ban đầu, hàm f(Tk) sẽ có giá trị là

Fk=2+1+3+1+0+1+2+2=12

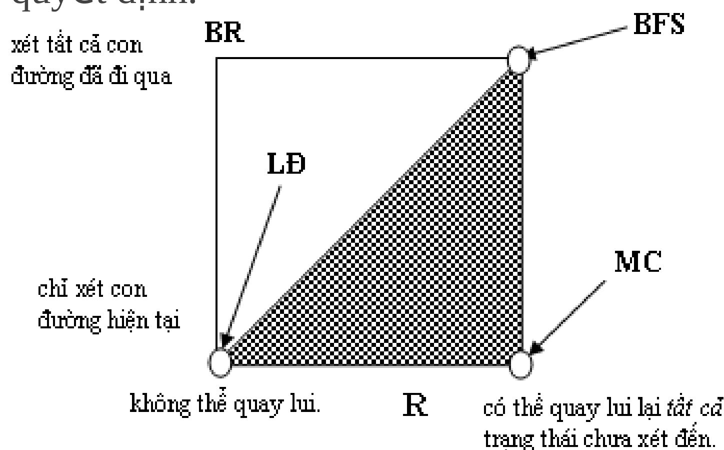
III.11. Các chiến lược tìm kiếm lai

Chúng ta đã biết qua 4 kiểu tìm kiếm : leo đèo (LĐ), tìm theo chiều sâu (MC), tìm theo chiều rộng (BR) và tìm kiếm BFS. Bốn kiểu tìm kiếm này có thể được xem như 4 thái cực của không gian liên tục bao gồm các chiến lược tìm kiếm khác nhau. Để giải thích điều này rõ hơn, sẽ tiện

hơn cho chúng ta nếu nhìn một chiến lược tìm kiếm lời giải dưới hai chiều sau :

Chiều khả năng quay lui (R): là khả năng cho phép quay lại để xem xét những trạng thái xét đến trước đó nếu gặp một trạng thái không thể đi tiếp.

Chiều phạm vi của sự đánh giá (S): số các trạng thái xét đến trong mỗi quyết định.

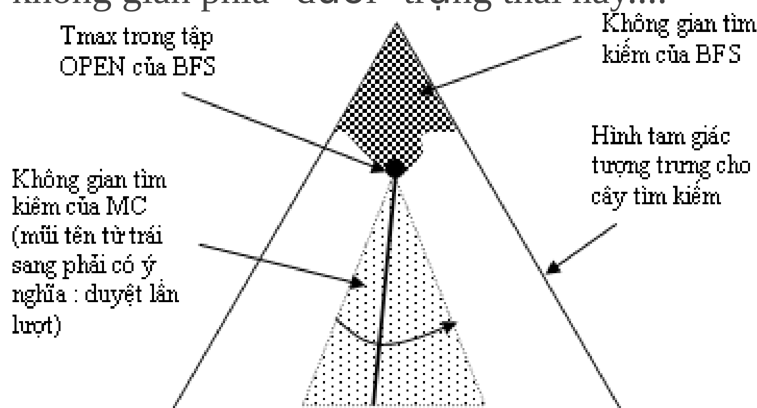


Hình : Tương quan giữa các chiến lược leo đèo, quay lui và tốt nhất

Theo hướng R, chúng ta thấy leo đèo nằm ở một thái cực (nó không cho phép quay lại những trạng thái chưa được xét đến), trong khi đó tìm kiếm quay lui và BFS ở một thái cực khác (cho phép quay lại tất cả các hướng đi chưa xét đến). Theo hướng S chúng ta thấy leo đèo và lần ngược nằm ở một thái cực (chỉ tập trung vào một phạm vi hẹp trên tập các trạng thái mới tạo ra từ trạng thái hiện tại) và BFS nằm ở một thái cực khác (trong khi BF xem xét toàn bộ tập các con đường đã có, bao gồm cả những con đường mới được tạo ra cũng như tất cả những con đường không được xét tới trước đây trước mỗi một quyết định).

Những thái cực này được trực quan hóa bằng hình ở trên. Vùng in đậm biểu diễn một mặt phẳng liên tục các chiến lược tìm kiếm mà nó kết hợp một số đặc điểm của một trong ba thái cực (leo đèo, chiều sâu, BFS) để có được một hòa hợp các đặc tính tính toán của chúng.

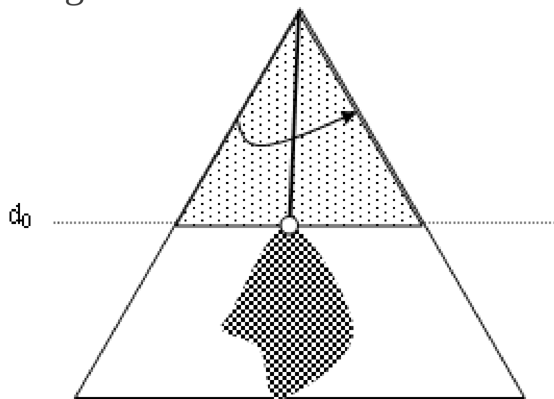
Nếu chúng ta không đủ bộ nhớ cần thiết để áp dụng thuật toán BFS thuần túy. Ta có thể kết hợp BFS với tìm theo chiều sâu để giảm bớt yêu cầu bộ nhớ. Dĩ nhiên, cái giá mà ta phải trả là số lượng các trạng thái có thể xét đến tại một bước sẽ nhỏ đi. Một loại kết hợp như thế được chỉ ra trong hình dưới. Trong hình này, thuật giải BFS được áp dụng tại đỉnh của đồ thị tìm kiếm (biểu diễn bằng vùng tô tậm) và tìm kiếm theo chiều sâu được áp dụng tại đáy (biểu diễn bởi tam giác tô nhạt). Đầu tiên ta áp dụng BFS vào trạng thái ban đầu T0 một cách bình thường. BFS sẽ thi hành cho đến một lúc nào đó, số lượng trạng thái được lưu trữ chiếm dụng một không gian bộ nhớ vượt quá một mức cho phép nào đó. Đến lúc này, ta sẽ áp dụng tìm kiếm chiều sâu xuất phát từ trạng thái tốt nhất Tmax trong OPEN cho tới khi toàn bộ không gian con phía "dưới" trạng thái đó được duyệt hết. Nếu không tìm thấy kết quả, trạng thái Tmax này được ghi nhận là không dẫn đến kết quả và ta lại chọn ra trạng thái tốt thứ hai trong OPEN và lại áp dụng tìm kiếm chiều sâu cho cho phần không gian phía "dưới" trạng thái này....



Hình : Chiến lược lai BFS-MC trong đó, BFS áp dụng tại đỉnh và MC tại đáy.

Một cách kết hợp khác là dùng tìm kiếm chiều sâu tại đỉnh không gian tìm kiếm và BFS được dùng tại đáy. Chúng ta áp dụng tìm kiếm chiều sâu cho tới khi gặp một trạng thái Tk mà độ sâu (số trạng thái trung gian) của nó vượt quá một ngưỡng d0 nào đó. Tại điểm này, thay vì lần ngược trở lại, ta áp dụng kiểu tìm kiếm BFS cho phần không gian phía "dưới" bắt đầu từ Tk cho tới khi nó trả về một giải pháp hoặc không tìm thấy. Nếu nó không tìm thấy kết quả, chúng ta lần ngược trở lại và lại dùng

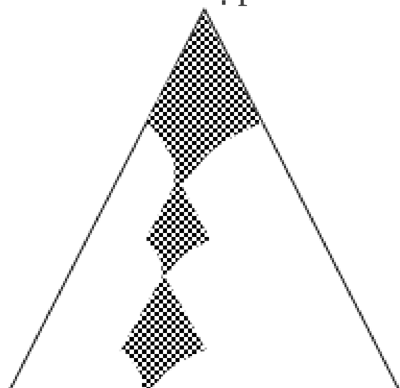
BFS khi đạt độ sâu d0. Tham số d0 sẽ được chọn sao cho bộ nhớ dùng cho tìm kiếm BFS trên không gian "dưới" mức d0 sẽ không vượt quá một hằng số cho trước. Rõ ràng ta ta không dễ gì xác định được d0 (vì nói chung, ta khó đánh giá được không gian bài toán rộng đến mức nào). Tuy nhiên, kiểu kết hợp này lại có một thuận lợi. Phần đáy không gian tìm kiếm thường chứa nhiều thông tin "bổ ích" hơn là phần đỉnh. (Chẳng hạn, tìm đường đi đến khu trung tâm của thành phố, khi càng đến gần khu trung tâm – đáy đồ thị – bạn càng dễ dàng tiến đến trung tâm hơn vì có nhiều "dấu hiệu" của trung tâm xuất hiện xung quanh bạn!). Nghĩa là, càng tiến về phía đáy của không gian tìm kiếm, ước lượng h’ thường càng trở nên chính xác hơn và do đó, càng dễ dẫn ta đến kết quả hơn.



Hình : Chiến lược lai BFS-MC trong đó, MC áp dụng tại đỉnh và BFS tại đáy.

Còn một kiểu kết hợp phức tạp hơn nữa. Trong đó, BFS được thực hiện cục bộ và chiều sâu được thực hiện toàn cục. Ta bắt đầu tìm kiếm theo BFS cho tới khi một sự lượng bộ nhớ xác định M0 được dùng hết. Tại điểm này, chúng ta xem tất cả những trạng thái trong OPEN như những trạng thái con trực tiếp của trạng thái ban đầu và chuyển giao chúng cho tìm kiếm chiều sâu. Tìm kiếm chiều sâu sẽ chọn trạng thái tốt nhất trong những trạng thái con này và "bành trướng" nó dùng BFS, nghĩa là nó chuyển trạng thái đã chọn cho tìm kiếm BFS cục bộ cho đến khi một lượng bộ nhớ M0 lại được dùng hết và trạng thái con mới trong OPEN lại tiếp tục được xem như nút con của nút "bành trướng"...Nếu việc

"bành trướng" bằng BFS thất bại thì ta quay lui lại và chọn nút con tốt thứ hai của tập OPEN trước đó, rồi lại tiếp tục bành trướng bằng BFS...



Hình : Chiến lược lai BFS-MC trong đó, BFS được áp dụng cục bộ và chiều sâu được áp dụng toàn cục.

Có một cách phối hợp nổi tiếng khác được gọi là tìm kiếm theo giai đoạn được thực hiện như sau. Thay vì lưu trữ trong bộ nhớ toàn bộ cây tìm kiếm được sinh ra bởi BFS, ta chỉ giữ lại cây con có triển vọng nhất. Khi một lượng bộ nhớ M0 được dùng hết, ta sẽ đánh dấu một tập con các trạng thái trong OPEN (những trạng thái có giá trị hàm f thấp nhất) để giữ lại; những đường đi tốt nhất qua những trạng thái này cũng sẽ được ghi nhớ và tất cả phần còn lại của cây bị loại bỏ. Quá trình tìm kiếm sau đó sẽ tiếp tục theo BFS cho tới khi một lượng bộ nhớ M0 lại được dùng hết và cứ thế. Chiến lược này có thể được xem như là một sự lai ghép giữa BF và leo đèo. Trong đó, leo đèo thuần túy loại bỏ tất cả nhưng chỉ giữ lại phương án tốt nhất còn tìm kiếm theo giai đoạn loại bỏ tất cả nhưng chỉ giữ lại tập các phương án tốt nhất.

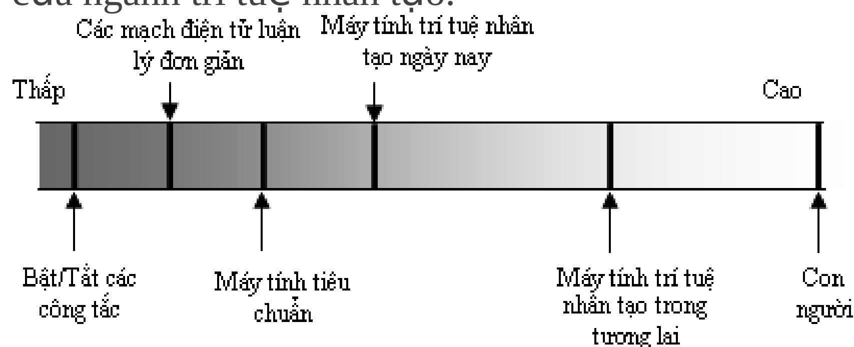
A. TỔNG QUAN TRÍ TUỆ NHÂN TẠO

I. MỞ ĐẦU

Chế tạo được những cỗ máy thông minh như con người (thậm chí thông minh hơn con người) là một ước mơ cháy bỏng của loài người từ hàng ngàn năm nay. Hẳn bạn đọc còn nhớ đến nhà khoa học Alan Turing cùng

những đóng góp to lớn của ông trong lĩnh vực trí tuệ nhân tạo. Năng lực máy tính ngày càng mạnh mẽ là một điều kiện hết sức thuận lợi cho trí tuệ nhân tạo. Điều này cho phép những chương trình máy tính áp dụng các thuật giải trí tuệ nhân tạo có khả năng phản ứng nhanh và hiệu quả hơn trước. Sự kiện máy tính Deep Blue đánh bại kiện tướng cờ vua thế giới Casparov là một minh chứng hùng hồn cho một bước tiến dài trong công cuộc nghiên cứu về trí tuệ nhân tạo. Tuycó thể đánh bại được Casparov nhưng Deep Blue là một cỗ máy chỉ biết đánh cờ ! Nó thậm chí không có được trí thông minh sơ đẳng của một đứa bé biết lên ba như nhận diện được những người thân, khả năng quan sát nhận biết thế giới, tình cảm thương, ghét, ... Ngành trí tuệ nhân tạo đã có những bước tiến đáng kể, nhưng một trí tuệ nhân tạo thực sự vẫn chỉ có trong những bộ phim khoa học giả tưởng của Hollywood. Vậy thì tại sao chúng ta vẫn nghiên cứu về trí tuệ nhân tạo? Điều này cũng tương tự như ước mơ chế tạo vàng của các nhà giả kim thuật thời Trung Cổ, tuy chưa thành công nhưng chính quá trình nghiên cứu đã làm sáng tỏ nhiều vấn đề.

Mặc dù mục tiêu tối thượng của ngành TTNT là xây dựng một chiếc máy có năng lực tư duy tương tự như con người nhưng khả năng hiện tại của tất cả các sản phẩm TTNT vẫn còn rất khiêm tốn so với mục tiêu đã đề ra. Tuy vậy, ngành khoa học mới mẻ này vẫn đang tiến bộ mỗi ngày và đang tỏ ra ngày càng hữu dụng trong một số công việc đòi hỏi trí thông minh của con người. Hình ảnh sau sẽ giúp bạn hình dung được tình hình của ngành trí tuệ nhân tạo.



Trước khi bước vào tìm hiểu về trí tuệ nhân tạo, chúng ta hãy nhắc lại một định nghĩa được nhiều nhà khoa học chấp nhận.



Mục tiêu của ngành khoa học trí tuệ nhân tạo ?

Tạo ra những chiếc máy tính có khả năng nhận thức, suy luận và phản ứng.

Nhận thức được hiểu là khả năng quan sát, học hỏi, hiểu biết cũng như những kinh nghiệm về thế giới xung quanh. Quá trình nhận thức giúp con người có tri thức. Suy luận là khả năng vận dụng những tri thức sẵn có để phản ứng với những tình huống hay những vấn đề - bài toán gặp phải trong cuộc sống. Nhận thức và suy luận để từ đó đưa ra những phản ứng thích hợp là ba hành vi có thể nói là đặc trưng cho trí tuệ của con người. (Dĩ nhiên còn một yếu tố nữa là tình cảm. Nhưng chúng ta sẽ không đề cập đến ở đây!). Do đó, cũng không có gì ngạc nhiên khi muốn tạo ra một chiếc máy tính thông minh, ta cần phải trang bị cho nó những khả năng này. Cả ba khả năng này đều cần đến một yếu tố cơ bản là tri thức.

Dưới góc nhìn của tập sách này, xây dựng trí tuệ nhân tạo là tìm cách biểu diễn tri thức, tìm cách vận dụng tri thức để giải quyết vấn đề và tìm cách bổ sung tri thức bằng cách "phát hiện" tri thức từ các thông tin sẵn có (máy học).

II. THÔNG TIN, DỮ LIỆU VÀ TRI THỨC

Tri thức là một khái niệm rất trừu tượng. Do đó, chúng ta sẽ không cố gắng đưa ra một định nghĩa hình thức chính xác ở đây. Thay vào đó, chúng ta hãy cùng nhau cảm nhận khái niệm "tri thức" bằng cách so sánh nó với hai khái niệm khác là thông tin và dữ liệu.

Nhà bác học nổi tiếng Karan Sing đã từng nói rằng "Chúng ta đang ngập chìm trong biển thông tin nhưng lại đang khát tri thức". Câu nói này làm nổi bật sự khác biệt về lượng lẫn về chất giữa hai khái niệm thông tin và tri thức.

Trong ngữ cảnh của ngành khoa học máy tính, người ta quan niệm rằng dữ liệu là các con số, chữ cái, hình ảnh, âm thanh... mà máy tính có thể

tiếp nhận và xử lý. Bản thân dữ liệu thường không có ý nghĩa đối với con người. Còn thông tin là tất cả những gì mà con người có thể cảm nhận được một cách trực tiếp thông qua các giác quan của mình (khứu giác, vị giác, thính giác, xúc giác, thị giác và giác quan thứ 6) hoặc gián tiếp thông qua các phương tiện kỹ thuật như tivi, radio, cassette,... Thông tin đối với con người luôn có một ý nghĩa nhất định nào đó. Với phương tiện máy tính (mà cụ thể là các thiết bị đầu ra), con người sẽ tiếp thu được một phần dữ liệu có ý nghĩa đối với mình. Nếu so về lượng, dữ liệu thường nhiều hơn thông tin.

Cũng có thể quan niệm thông tin là quan hệ giữa các dữ liệu. Các dữ liệu được sắp xếp theo một thứ tự hoặc được tập hợp lại theo một quan hệ nào đó sẽ chứa đựng thông tin. Nếu những quan hệ này được chỉ ra một cách rõ ràng thì đó là các tri thức. Chẳng hạn :



Trong toán học :

Bản thân từng con số riêng lẻ như 1, 1, 3, 5, 2, 7, 11, ... là các dữ liệu. Tuy nhiên, khi đặt chúng lại với nhau theo trật tự như dưới đây thì giữa chúng đã bắt đầu có một mối liên hệ

Dữ liệu : 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Mối liên hệ này có thể được biểu diễn bằng công thức sau : Un = Un-1 + Un-2.

Công thức nêu trên chính là tri thức.



Trong vật lý :

Bản sau đây cho chúng ta biết số đo về điện trở (R), điện thế (U) và cường độ dòng điện (I) trong một mạch điện.

I	U	R
5	10	2
2.5	20	8
4	12	3
7.3	14.6	2

Bản thân những con số trong các cột của bản trên không có mấy ý nghĩa nếu ta tách rời chúng ta. Nhưng khi đặt kế nhau, chúng đã cho thấy có một sự liên hệ nào đó. Và mối liên hệ này có thể được diễn tả bằng công thức đơn giản sau :

$$I = \frac{U}{R}$$

Công thức này là tri thức.



Trong cuộc sống hàng ngày :

Hằng ngày, người nông dân vẫn quan sát thấy các hiện tượng nắng, mưa, râm và chuồn chuồn bay. Rất nhiều lần quan sát, họ đã có nhận xét như sau :

Chuồn chuồn bay thấp thì mưa, bay cao thì nắng, bay vừa thì râm.

Lời nhận xét trên là tri thức.

Có quan điểm trên cho rằng chỉ những mối liên hệ tường minh (có thể chứng minh được) giữa các dữ liệu mới được xem là tri thức. Còn những mối quan hệ không tường minh thì không được công nhận. Ở đây, ta cũng có thể quan niệm rằng, mọi mối liên hệ giữa các dữ liệu đều có thể

được xem là tri thức, bởi vì, những mối liên hệ này thực sự tồn tại. Điểm khác biệt là chúng ta chưa phát hiện ra nó mà thôi. Rõ ràng rằng "dù sao thì trái đất cũng vẫn xoay quanh mặt trời" dù tri thức này có được Galilê phát hiện ra hay không!

Như vậy, so với dữ liệu thì tri thức có số lượng ít hơn rất nhiều. Thuật ngữ ít ở đây không chỉ đơn giản là một dấu nhỏ hơn bình thường mà là sự kết tinh hoặc cô đọng lại. Bạn hãy hình dung dữ liệu như là những điểm trên mặt phẳng còn tri thức chính là phương trình của đường cong nối tất cả những điểm này lại. Chỉ cần một phương trình đường cong ta có thể biểu diễn được vô số điểm!. Cũng vậy, chúng ta cần có những kinh nghiệm, nhận xét từ hàng đống số liệu thống kê, nếu không, chúng ta sẽ ngập chìm trong biển thông tin như nhà bác học Karan Sing đã cảnh báo!.

Người ta thường phân loại tri thức ra làm các dạng như sau :



Tri thức sự kiện : là các khẳng định về một sự kiện, khái niệm nào đó (trong một phạm vi xác định). Các định luật vật lý, toán học, ... thường được xếp vào loại này. (Chẳng hạn : mặt trời mọc ở đằng đông, tam giác đều có 3 góc 600, ...)



Tri thức thủ tục : thường dùng để diễn tả phương pháp, các bước cần tiến hành, trình từ hay ngắn gọn là cách giải quyết một vấn đề. Thuật toán, thuật giải là một dạng của tri thức thủ tục.



Tri thức mô tả : cho biết một đối tượng, sự kiện, vấn đề, khái niệm, ... được thấy, cảm nhận, cấu tạo như thế nào (một cái bàn thường có 4 chân, con người có 2 tay, 2 mắt,...)



Tri thức Heuristic : là một dạng tri thức cảm tính. Các tri thức thuộc loại này thường có dạng ước lượng, phỏng đoán, và thường được hình thành thông qua kinh nghiệm.

Trên thực tế, rất hiếm có một trí tuệ mà không cần đến tri thức (liệu có thể có một đại kiện tướng cờ vua mà không biết đánh cờ hoặc không biết các thế cờ quan trọng không?). Tuy tri thức không quyết định sự thông minh (người biết nhiều định lý toán hơn chưa chắc đã giải toán giỏi hơn!) nhưng nó là một yếu tố cơ bản cấu thành trí thông minh. Chính vì vậy, muốn xây dựng một trí thông minh nhân tạo, ta cần phải có yếu tố cơ bản này. Từ đây đặt ra vấn đề đầu tiên là … Các phương pháp đưa tri thức vào máy tính được gọi là biểu diễn tri thức.

III. THUẬT TOÁN – MỘT PHƯƠNG PHÁP BIỄU DIỄN TRI THỨC?

Trước khi trả lời câu hỏi trên, bạn hãy thử nghĩ xem, liệu một chương trình giải phương trình bậc 2 có thể được xem là một chương trình có tri thức hay không? ... Có chứ ! Vậy thì tri thức nằm ở đâu? Tri thức về giải phương trình bậc hai thực chất đã được mã hóa dưới dạng các câu lệnh if..then..else trong chương trình. Một cách tổng quát, có thể khẳng định là tất cả các chương trình máy tính ít nhiều đều đã có tri thức. Đó chính là tri thức của lập trình viên được chuyển thành các câu lệnh của chương trình. Bạn sẽ thắc mắc "như vậy tại sao đưa tri thức vào máy tính lại là một vấn đề ? (vì từ trước tới giờ chúng ta đã, đang và sẽ tiếp tục làm như thế mà?)". Đúng như thế thật, nhưng vấn đề nằm ở chỗ, các tri thức trong những chương trình truyền thống là những tri thức "cứng", nghĩa là nó không thể được thêm vào hay điều chỉnh một khi chương trình đã được biên dịch. Muốn điều chỉnh thì chúng ta phải tiến hành sửa lại mã nguồn của chương trình (rồi sau đó biên dịch lại). Mà thao tác sửa chương trình thì chỉ có những lập trình viên mới có thể làm được. Điều này sẽ làm giảm khả năng ứng dụng chương trình (vì đa số người dùng bình thường đều không biết lập trình).

Bạn thử nghĩ xem, với một chương trình hỗ trợ ra quyết định (như đầu tư cổ phiếu, đầu tư bất động sản chẳng hạn), liệu người dùng có cảm thấy thoải mái không khi muốn đưa vào chương trình những kiến thức

của mình thì anh ta phải chọn một trong hai cách là (1) tự sửa lại mã chương trình!? (2) tìm tác giả của chương trình để nhờ người này sửa lại!?. Cả hai thao tác trên đều không thể chấp nhận được đối với bất kỳ người dùng bình thường nào. Họ cần có một cách nào đó để chính họ có thể đưa tri thức vào máy tính một cách dễ dàng, thuận tiện giống như họ đang đối thoại với một con người.

Để làm được điều này, chúng ta cần phải "mềm" hóa các tri thức được biểu diễn trong máy tính. Xét cho cùng, mọi chương trình máy tính đều gồm hai thành phần là các mã lệnh và dữ liệu. Mã lệnh được ví như là phần cứng của chương trình còn dữ liệu được xem là phần mềm (vì nó có thể được thay đổi bởi người dùng). Do đó, "mềm" hóa tri thức cũng đồng nghĩa với việc tìm các phương pháp để có thể biểu diễn các loại tri thức của con người bằng các cấu trúc dữ liệu mà máy tính có thể xử lý được. Đây cũng chính là ý nghĩa của thuật ngữ "biểu diễn tri thức".

Bạn cần phải biết rằng, ít ra là cho đến thời điểm bạn đang đọc cuốn sách này, con người vẫn chưa thể tìm ra một kiểu biểu diễn tổng quát cho mọi loại tri thức!

Để làm vấn đề mà chúng ta đang bàn luận trở nên sáng tỏ hơn. Chúng ta hãy xem xét một số bài toán trong phần tiếp theo.

IV. LÀM QUEN VỚI CÁCH GIẢI QUYẾT VẤN ĐỀ BẰNG CÁCH CHUYỂN GIAO TRI THỨC CHO MÁY TÍNH



Bài toán 1 : Cho hai bình rỗng X và Y có thể tích lần lượt là VX và VY, hãy dùng hai bình này để đong ra z lít nước (z <= min(VX,VY)).



Bài toán 2 : Cho biết một số yếu tố của tam giác (như chiều dài cạnh và góc, ...). Hãy tính các yếu tố còn lại.



Bài toán 3 : Tính diện tích phần giao của các hình hình học cơ bản.

Hai bài toán đầu là hai bài toán khá tiêu biểu, thường được dùng để minh họa cho nét đẹp của phương pháp giải quyết vấn đề bài toán bằng cách chuyển giao tri thức cho máy tính. Nếu sử dụng thuật toán thông thường, chúng ta thường chỉ giải được một số trường hợp cụ thể của các bài toán này. Thậm chí, nhiều người khi mới tiếp cận với 2 bài toán này còn không tin là nó có thể hoàn toàn được giải một cách tổng quát bởi máy tính!. Bài toán số 3 là một minh họa đẹp mắt cho kỹ thuật giải quyết vấn đề "vĩ mô", nghĩa là ta chỉ cần mô tả các bước giải quyết ở mức tổng quát cho máy tính mà không cần đi vào cài đặt cụ thể.

Bài toán 1 sẽ được giải quyết bằng cách sử dụng các luật dẫn xuất (luật sinh). Bài toán 2 sẽ được giải quyết bằng mạng ngữ nghĩa và bài toán 3 sẽ giải quyết bằng công cụ frame. Ở đây chúng ta cùng nhau tìm hiểu cách giải bài toán đầu tiên. Hai bài toán kế tiếp sẽ được giải quyết lần lượt ở các mục sau.

Với một trường hợp cụ thể của bài toán 1, như VX = 5 và VY = 7 và z = 4. Sau một thời gian tính toán, bạn có thể sẽ đưa ra một quy trình đổ nước đại loại như :

- Múc đầy bình 7

- Trút hết qua bình 5 cho đến khi 5 đầy.

- Đổ hết nước trong bình 5

- Đổ hết nước còn lại từ bình 7 sang bình 5

- Múc đầy bình 7

-

Trút hết qua bình 5 cho đến khi bình 5 đầy.

Phần còn lại chính là số nước cần đong.

Tuy nhiên, với những số liệu khác, bạn phải "mày mò" lại từ đầu để tìm ra quy trình đổ nước. Cứ thế, mỗi một trường hợp sẽ có một cách đổ nước hoàn toàn khác nhau. Như vậy, nếu có một ai đó yêu cầu bạn đưa ra một cách làm tổng quát thì chính bạn cũng sẽ lúng túng (dĩ nhiên, ngoại trừ trường hợp bạn đã biết trước cách giải theo tri thức mà chúng ta sắp sửa tìm hiểu ở đây!).

Đến đây, bạn hãy bình tâm kiểm lại cách thức bạn tìm kiếm lời giải cho một trường hợp cụ thể. Vì chưa tìm ra một quy tắc cụ thể nào, bạn sẽ thực hiện một loạt các thao tác "cảm tính" như đong đầy một bình, trút một bình này sang bình kia, đổ hết nước trong một bình ra... vừa làm vừa nhẩm tính xem cách làm này có thể đi đến kết quả hay không. Sau nhiều lần thí nghiệm, rất có thể bạn sẽ rút ra được một số kinh nghiệm như "khi bình 7 đầy nước mà bình 5 chưa đầy thì hãy đổ nó sang bình 5 cho đến khi bình 5 đầy"... Vậy thì tại sao bạn lại không thử "truyền" những kinh nghiệm này cho máy tính và để cho máy tính "mày mò" tìm các thao tác cho chúng ta? Điều này hoàn toàn có lợi, vì máy tính có khả năng "mày mò" hơn hẳn chúng ta! Nếu những "kinh nghiệm" mà chúng ta cung cấp cho máy tính không giúp chúng ta tìm được lời giải, chúng ta sẽ thay thế nó bằng những kinh nghiệm khác và lại tiếp tục để máy tính tìm kiếm lời giải!

Chúng ta hãy phát biểu lại bài toán một cách hình thức hơn.

Không làm mất tính tổng quát, ta luôn có thể giả sử rằng VX<VY.

Gọi lượng nước chứa trong bình X là x (0<=x<=VX)

Gọi lượng nước chứa trong bình Y là y (0<=y<=VY)

Như vậy, điều kiện kết thúc của bài toán sẽ là :

x = z hoặc y = z

Điều kiện đầu của bài toán là : x = 0 và y=0

Quá trình giải được thực hiện bằng cách xét lần lượt các luật sau, luật nào thỏa mãn thì sẽ được áp dụng. Lúc này, các luật chính là các "kinh nghiệm" hay tri thức mà ta đã chuyển giao cho máy tính. Sau khi áp dụng luật, trạng thái của bài toán sẽ thay đổi, ta lại tiếp tục xét các luật kế tiếp, nếu hết luật, quay trở lại luật đầu tiên. Quá trình tiếp diễn cho đến khi đạt được điều kiện kết thúc của bài toán.

Ba luật này được mô tả như sau :

(L1) Nếu bình X đầy thì đổ hết nước trong bình X đi.

(L2) Nếu bình Y rỗng thì đổ đầy nước vào bình Y.

(L3) Nếu bình X không đầy và bình Y không rỗng thì hãy trút nước t? bình Y sang bình X (cho đến khi bình X đầy hoặc bình Y hết nước).

Trên thực tế, lúc đầu để giải trường hợp tổng quát của bài toán này, người ta đã dùng đến hơn 15 luật (kinh nghiệm) khác nhau. Tuy nhiên, sau này, người ta đã rút gọn lại chỉ còn 3 luật như trên.

Bạn có thể dễ dàng chuyển đổi cách giải này thành chương trình như sau :

```
...

x := 0; y := 0;

WHILE ( (x <> z) AND (y<>z) ) DO BEGIN

IF (x = Vx) THEN x := 0;

IF (y = 0) THEN (y:= Vy);

IF (y > 0) THEN BEGIN
```

k:= min(Vx - x, y);

x := x + k;

y := y - k;

END;

END;

...

Thử "chạy" chương trình trên với số liệu cụ thể là :

Vx = 3, Vy = 4 và z = 2

Ban đầu : x = 0, y = 0

Luật (L2) -> x = 0, y = 4

Luật (L3) -> x = 3, y = 1

Luật (L1) -> x = 0, y = 1

Luật (L3) -> x = 1, y = 0

Luật (L2) -> x = 1, y = 4

Luật (L3) -> x = 3, y = 2

3 luật mà chúng ta đã cài đặt trong chương trình ở trên được gọi là cơ sở tri thức. Còn cách thức tìm kiếm lời giải bằng cách duyệt tuần tự từng luật và áp dụng nó được gọi là động cơ suy diễn. Chúng ta sẽ định nghĩa chính xác hai thuật ngữ này ở cuối mục.

Người ta đã chứng minh được rằng, bài toán đong nước chỉ có lời giải khi số nước cần đong là một bội số của ước số chung lớn nhất của thể tích hai bình.

z = n   USCLN(VX, VY) (với n nguyên dương)

Cách giải quyết vấn đề theo kiểu này khác so với cách giải bằng thuật toán thông thường là chúng ta không đưa ra một trình tự giải quyết vấn đề cụ thể mà chỉ đưa ra các quy tắc chung chung (dưới dạng các luật), máy tính sẽ dựa vào đó (áp dụng các luật) để tự xây dựng một quy trình giải quyết vấn đề. Điều này cũng giống như việc chúng ta giải toán bằng cách đưa ra các định lý, quy tắc liên quan đến bài toán mà không cần phải chỉ ra cách giải cụ thể.

Vậy thì điểm thú vị nằm ở điểm nào? Bạn sẽ có thể cảm thấy rằng chúng ta vẫn đang dùng tri thức "cứng" ! (vì các tri thức vẫn là các câu lệnh IF được cài sẵn trong chương trình). Thực ra thì chương trình của chúng ta đã "mềm" hơn một tí rồi đấy. Nếu không tin, các bạn hãy quan sát phiên bản kế tiếp của chương trình này.

```
FUNCTION DK(L INTEGER):BOOLEAN;

BEGIN

CASE L OF

1 : DK := (x = Vx);

2 : DK := (y = 0);

3 : DK := (y>0);

END;

END;

PROCEDURE ThiHanh(L INTEGER):BOOLEAN;

BEGIN

CASE L OF
```

```
1 : x := 0;

2: y := Vy;

3 : BEGIN

k := min(Vx-x,y);

x := x+k;

y := y-k;

END;

END;

END;

CONST SO_LUAT = 3;

BEGIN

WHILE (x<>z) AND (y<>z) DO BEGIN

FOR i:=1 TO SO_LUAT DO

IF DK(L) THEN ThiHanh(L);

END;

END.
```

Đoạn chương trình chính cũng thi hành bằng cách lần lượt xét qua 3 lệnh IF như chương trình đầu tiên. Tuy nhiên, ở đây, biểu thức điều kiện được thay thế bằng hàm DK và các hành động ứng với điều kiện đã được thay thế bằng thủ tục ThiHanh. Tính chất "mềm" hơn của chương trình này thể hiện ở chỗ, nếu muốn bổ sung "tri thức", ta chỉ phải điều chỉnh lại các hàm DK và ThiHanh mà không cần phải sửa lại chương trình chính.

Bây giờ hãy giả sử rằng ta đã có hàm và thủ tục đặc biệt sau :

FUNCTION GiaTriBool(DK : String) : BOOLEAN;

PROCEDURE ThucHien(ThaoTac : String) ;

hàm GiaTriBool nhận vào một chuỗi điều kiện, nó sẽ phân tích chuỗi, tính toán rồi trả ra giá trị BOOLEAN của biểu thức này.

Ví dụ : GiaTriBoolean('6<7') sẽ trả ra FALSE

Thủ tục ThucHien cũng nhận vào một chuỗi, nó cũng sẽ phân tích chuỗi rồi tiến hành thực hiện những hành động được miêu tả trong chuỗi này.

Với hàm và thủ tục này, chương trình của chúng ta sẽ như sau :

CONST SO_LUAT = 3;

TYPE

Luat RECORD

DK : String;

ThiHanh : String;

END;

DSLuat ARRAY [1..SO_LUAT] OF Luat; 9;

VAR

CacLuat DSLuat;

PROCEDURE KhoiDong;

BEGIN

CacLuat[1].DK := 'x = Vx';

```
CacLuat[2].DK := ‘y = 0’;

CacLuat[3].DK := ‘y>0’; 9;

CacLuat[1].ThaoTac := ‘x:=0’;

CacLuat[2].ThaoTac:= ‘y:=Vy’;

CacLuat[3].ThaoTac:= ‘k:=min(Vx-x,y), x:=x+k, y:=y-k’;

END;

BEGIN

WHILE (x<>z) AND (y<>z) DO BEGIN

FOR i:=1 TO SO_LUAT DO

IF GiaTriBoolean(CacLuat[i].DK)

THEN ThucHien(CacLuat[i].ThaoTac);

END;

END.
```

Chúng ta tạm cho rằng trong quá trình chương trình thi hành, ta có thể dễ dàng thay đổi số phần tử mảng CacLuat (các ngôn ngữ lập trình sau này như Visual C++, Delphi đều cho phép điều này). Với chương trình này, khi muốn sửa đổi "tri thức", bạn chỉ cần thay đổi giá trị mảng Luat là xong.

Tuy nhiên, người dùng vẫn gặp khó khăn khi muốn bổ sung hoặc hiệu chỉnh tri thức. Họ cần phải nhập các chuỗi đại loại như ‘x=0’ hoặc ‘k:=min(Vx-x,y)’ ...Các chuỗi này, tuy có ý nghĩa đối với chương trình nhưng vẫn còn khá xa lạ đối với người dùng bình thường. Chúng ta cần giảm bớt "khoảng cách" này lại bằng cách đưa ra những chuỗi điều kiện hoặc thao tác có ý nghĩa trực tiếp đối với người dùng. Chương trình sẽ có

chuyển đổi lại các điều kiện và thao tác này sang dạng phù hợp với chương trình.

Để làm được điều trên. Chúng ta cần phải liệt kê được các trạng thái và thao tác cơ bản của bài toán này. Sau đây là một số trạng thái và thao tác cơ bản.



Trạng thái cơ bản :

Bình X đầy, Bình X rỗng, Bình X không rỗng, Bình X có n lít nước.



Thao tác

Đổ hết nước trong bình, Đổ đầy nước trong bình, Đổ nước từ bình A sang bình B cho đến khi B đầy hoặc A rỗng.

Lưu ý rằng ta không thể có thao tác "Đổ n lít nước từ A sang B" vì bài toán đã giả định rằng các bình đều không có vạch chia, hơn nữa nếu ta biết cách đổ n lít nước từ A sang B thì lời giải bài toán trở thành quá đơn giản.

"Múc đầy X"

"Đổ z lít nước từ X sang Y"

Vì đây là một bài toán đơn giản nên bạn có thể dễ nhận thấy rằng, các trạng thái cơ bản và thao tác chẳng có gì khác so với các điều kiện mà chúng ta đã đưa ra.

Kế tiếp, ta sẽ viết các đoạn chương trình cho phép người dùng nhập vào các luật (dạng nếu ... thì ...) được hình thành từ các trạng thái và điều kiện cơ bản này, đồng thời tiến hành chuyển sang dạng máy tính có thể xử lý được như ở ví dụ trên. Chúng ta sẽ không bàn đến việc cài đặt các đoạn chương trình giao tiếp với người dùng ở đây.

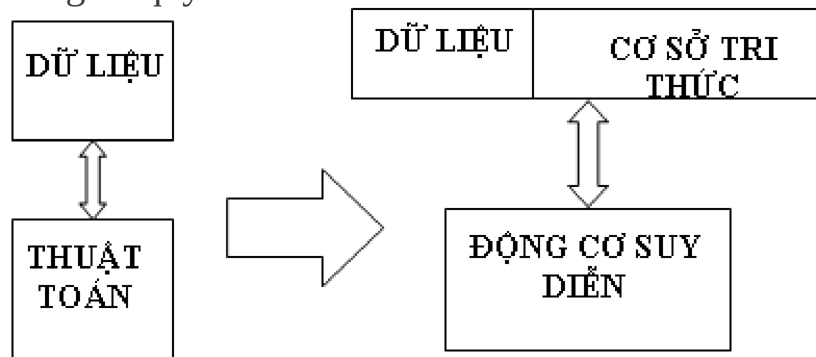
Như vậy, so với chương trình truyền thống (được cấu tạo từ hai "chất liệu" cơ bản là dữ liệu và thuật toán), chương trình trí tuệ nhân tạo được cấu tạo từ hai thành phần là cơ sở tri thức (knowledge base) và động cơ suy diễn (inference engine).



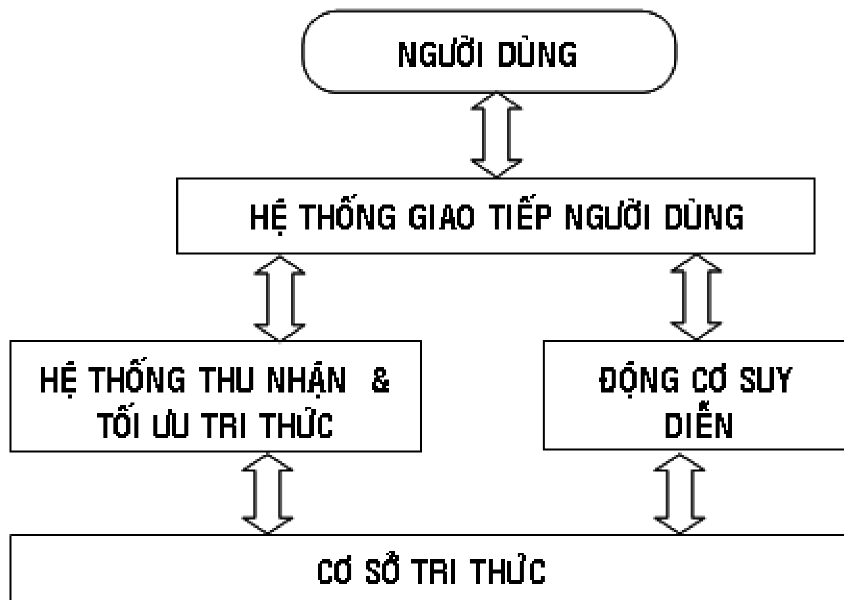
Cơ sở tri thức : là tập hợp các tri thức liên quan đến vấn đề mà chương trình quan tâm giải quyết.



Động cơ suy diễn : là phương pháp vận dụng tri thức trong cơ sở tri thức để giải quyết vấn đề.



Nếu xét theo quan niệm biểu diễn tri thức mà ta vừa bàn luận ở trên thì cơ sở tri thức chỉ là một dạng dữ liệu đặc biệt và động cơ suy diễn cũng chỉ là một dạng của thuật toán đặc biệt mà thôi. Tuy vậy, có thể nói rằng, cơ sở tri thức và động cơ suy diễn là một bước tiến hóa mới của dữ liệu và thuật toán của chương trình! Bạn có thể hình dung động cơ suy diễn giống như một loại động cơ tổng quát, được chuẩn hóa có thể dùng để vận hành nhiều loại xe máy khác nhau và cơ sở tri thức chính là loại nhiên liệu đặc biệt để vận hành loại động cơ này !



Cơ sở tri thức cũng gặp phải những vấn đề tương tự như những cơ sở dữ liệu khác như sự trùng lắp, thừa, mâu thuẫn. Khi xây dựng cơ sở tri thức, ta cũng phải chú ý đến những yếu tố này. Như vậy, bên cạnh vấn đề biểu diễn tri thức, ta còn phải đề ra các phương pháp để loại bỏ những tri thức trùng lắp, thừa hoặc mâu thuẫn. Những thao tác này sẽ được thực hiện trong quá trình ghi nhận tri thức vào hệ thống. Chúng ta sẽ đề cập đến những phương pháp này trong phần tìm hiểu về các luật dẫn.

Hình ảnh trên tóm tắt cho chúng ta thấy cấu trúc chung nhất của một chương trình trí tuệ nhân tạo.

B. CÁC PHƯƠNG PHÁP BIỂU DIỄN TRI THỨC TRÊN MÁY TÍNH

V. LOGIC MỆNH ĐỀ

Đây có lẽ là kiểu biểu diễn tri thức đơn giản nhất và gần gũi nhất đối với chúng ta. Mệnh đề là một khẳng định, một phát biểu mà giá trị của nó chỉ có thể hoặc là đúng hoặc là sai.

Ví dụ :

phát biểu "1+1=2" có giá trị đúng.

phát biểu "Mọi loại cá có thể sống trên bờ" có giá trị sai.

Giá trị của mệnh đề không chỉ phụ thuộc vào bản thân mệnh đề đó. Có những mệnh đề mà giá trị của nó luôn đúng hoặc sai bất chấp thời gian nhưng cũng có những mệnh đề mà giá trị của nó lại phụ thuộc vào thời gian, không gian và nhiều yếu tố khác quan khác. Chẳng hạn như mệnh đề : "Con người không thể nhảy cao hơn 5m với chân trần" là đúng khi ở trái đất , còn ở những hành tinh có lực hấp dẫn yếu thì có thể sai.

Ta ký hiệu mệnh đề bằng những chữ cái la tinh như a, b, c, ...

Có 3 phép nối cơ bản để tạo ra những mệnh đề mới từ những mệnh đề cơ sở là phép hội (   ), giao(   ) và phủ định (   )

Bạn đọc chắn hẳn đã từng sử dụng logic mệnh đề trong chương trình rất nhiều lần (như trong cấu trúc lệnh IF ... THEN ... ELSE) để biểu diễn các tri thức "cứng" trong máy tính !

Bên cạnh các thao tác tính ra giá trị các mệnh đề phức từ giá trị những mệnh đề con, chúng ta có được một cơ chế suy diễn như sau :

- Modus Ponens : Nếu mệnh đề A là đúng và mệnh đề A   B là đúng thì giá trị của B sẽ là đúng.

- Modus Tollens : Nếu mệnh đề A   B là đúng và mệnh đề B là sai thì giá trị của A sẽ là sai.

Các phép toán và suy luận trên mệnh đề đã được đề cập nhiều đến trong các tài liệu về toán nên chúng ta sẽ không đi vào chi tiết ở đây.

VI. LOGIC VỊ TỪ

Biểu diễn tri thức bằng mệnh đề gặp phải một trở ngại cơ bản là ta không thể can thiệp vào cấu trúc của một mệnh đề. Hay nói một cách khác là mệnh đề không có cấu trúc . Điều này làm hạn chế rất nhiều thao

tác suy luận . Do đó, người ta đã đưa vào khái niệm vị từ và lượng từ (- với mọi, - tồn tại) để tăng cường tính cấu trúc của một mệnh đề.

Trong logic vị từ, một mệnh đề được cấu tạo bởi hai thành phần là các đối tượng tri thức và mối liên hệ giữa chúng (gọi là vị từ). Các mệnh đề sẽ được biểu diễn dưới dạng :

Vị từ (<đối tượng 1>, <đối tượng 2>, …, <đối tượng n>)

Như vậy để biểu diễn vị của các trái cây, các mệnh đề sẽ được viết lại thành :

Cam có vị Ngọt Vị (Cam, Ngọt)

Cam có màu Xanh Màu (Cam, Xanh)

...

Kiểu biểu diễn này có hình thức tương tự như hàm trong các ngôn ngữ lập trình, các đối tượng tri thức chính là các tham số của hàm, giá trị mệnh đề chính là kết quả của hàm (thuộc kiểu BOOLEAN).

Với vị từ, ta có thể biểu diễn các tri thức dưới dạng các mệnh đề tổng quát, là những mệnh đề mà giá trị của nó được xác định thông qua các đối tượng tri thức cấu tạo nên nó.

Chẳng hạn tri thức : "A là bố của B nếu B là anh hoặc em của một người con của A" có thể được biểu diễn dưới dạng vị từ như sau :

Bố (A, B) = Tồn tại Z sao cho : Bố (A, Z) và (Anh(Z, B) hoặc Anh(B,Z))

Trong trường hợp này, mệnh đề Bố(A,B) là một mệnh đề tổng quát

Như vậy nếu ta có các mệnh đề cơ sở là :

a) Bố ("An", "Bình") có giá trị đúng (Anh là bố của Bình)

b) Anh("Tú", "Bình") có giá trị đúng (Tú là anh của Bình)

thì mệnh đề c) Bố ("An", "Tú") sẽ có giá trị là đúng. (An là bố của Tú).

Rõ ràng là nếu chỉ sử dụng logic mệnh đề thông thường thì ta sẽ không thể tìm được một mối liên hệ nào giữa c và a,b bằng các phép nối mệnh đề , , . Từ đó, ta cũng không thể tính ra được giá trị của mệnh đề c. Sở dĩ như vậy vì ta không thể thể hiện tường minh tri thức "(A là bố của B) nếu có Z sao cho (A là bố của Z) và (Z anh hoặc em C)" dưới dạng các mệnh đề thông thường. Chính đặc trưng của vị từ đã cho phép chúng ta thể hiện được các tri thức dạng tổng quát như trên.

Thêm một số ví dụ nữa để các bạn thấy rõ hơn khả năng của vị từ :

Câu cách ngôn "Không có vật gì là lớn nhất và không có vật gì là bé nhất!" có thể được biểu diễn dưới dạng vị từ như sau :

LớnHơn(x,y) = x>y

NhỏHơn(x,y) = x<y

x, y : LớnHơn(y,x) và x, y : NhỏHơn(y,x)

Câu châm ngôn "Gần mực thì đen, gần đèn thì sáng" được hiểu là "chơi với bạn xấu nào thì ta cũng sẽ thành người xấu" có thể được biểu diễn bằng vị từ như sau :

NgườiXấu (x) = y : Bạn(x,y) và NgườiXấu(y)

Công cụ vị từ đã được nghiên cứu và phát triển thành một ngôn ngữ lập trình đặc trưng cho trí tuệ nhân tạo. Đó là ngôn ngữ PROLOG. Phần đọc thêm của chương sẽ giới thiệu tổng quan với các bạn về ngôn ngữ này.

VII. MỘT SỐ THUẬT GIẢI LIÊN QUAN ĐẾN LOGIC MỆNH ĐỀ

Một trong những vấn đề khá quan trọng của logic mệnh đề là chứng minh tính đúng đắn của phép suy diễn (a b). Đây cũng chính là bài toán chứng minh thường gặp trong toán học.

Rõ ràng rằng với hai phép suy luận cơ bản của logic mệnh đề (Modus Ponens, Modus Tollens) cộng với các phép biến đổi hình thức, ta cũng có thể chứng minh được phép suy diễn. Tuy nhiên, thao tác biến đối hình thức là rất khó cài đặt được trên máy tính. Thậm chí điều này còn khó khăn với cả con người!

Với công cụ máy tính, bạn có thể cho rằng ta sẽ dễ dàng chứng minh được mọi bài toán bằng một phương pháp "thô bạo" là lập bảng chân trị . Tuy về lý thuyết, phương pháp lập bảng chân trị luôn cho được kết quả cuối cùng nhưng độ phức tạp của phương pháp này là quá lớn, O(2n) với n là số biến mệnh đề. Sau đây chúng ta sẽ nghiên cứu hai phương pháp chứng minh mệnh đề với độ phức tạp chỉ có O(n).

VII.1. Thuật giải Vương Hạo

B1 : Phát biểu lại giả thiết và kết luận của vấn đề theo dạng chuẩn sau :

GT1, GT2, ..., GTn KL1, KL2, ..., KLm

Trong đó các GTi và KLi là các mệnh đề được xây dựng từ các biến mệnh đề và 3 phép nối cơ bản : , ,

B2 : Chuyển vế các GTi và KLi có dạng phủ định.

Ví dụ :

p q, (r s), g, p r s, p

p q, p r, p (r s), g, s

B3 : Nếu GTi có phép thì thay thế phép bằng dấu ","

Nếu KLi có phép thì thay thế phép bằng dấu ","

Ví dụ :

p q, r (p s) q, s

p, q, r, p s q, s

B4 : Nếu GTi có phép thì tách thành hai dòng con.

Nếu ở KLi có phép thì tách thành hai dòng con.

Ví dụ :

p, p q q

p, p q p, q q

B5 : Một dòng được chứng minh nếu tồn tại chung một mệnh đề ở ở cả hai phía.

Ví dụ :

p, q q được chứng minh

p, p q p p, q

B6 :

a) Nếu một dòng không còn phép nối hoặc ở cả hai vế và ở 2 vế không có chung một biến mệnh đề thì dòng đó không được chứng minh.

b) Một vấn đề được chứng minh nếu tất cả dòng dẫn xuất từ dạng chuẩn ban đầu đều được chứng minh.

VII.2 Thuật giải Robinson

Thuật giải này hoạt động dựa trên phương pháp chứng minh phản chứng.

Phương pháp chứng minh phản chứng

Chứng minh phép suy luận (a b) là đúng (với a là giả thiết, b là kết luận).

Phản chứng : giả sử b sai suy ra b là đúng.

Bài toán được chứng minh nếu a đúng và   b đúng sinh ra một mâu thuẫn.

B1 : Phát biểu lại giả thiết và kết luận của vấn đề dưới dạng chuẩn như sau :

GT1, GT2, ...,GTn   KL1, KL2, .., KLm

Trong đó : GTi và KLj được xây dựng từ các biến mệnh đề và các phép toán :   ,   ,

B2 : Nếu GTi có phép   thì thay bằng dấu ","

Nếu KLi có phép   thì thay bằng dấu ","

B3 : Biến đổi dòng chuẩn ở B1 về thành danh sách mệnh đề như sau :

{ GT1, GT2, ..., GTn ,   KL1,   KL2, ...,   KLm }

B4 : Nếu trong danh sách mệnh đề ở bước 2 có 2 mệnh đề đối ngẫu nhau thì bài toán được chứng minh. Ngược lại thì chuyển sang B4. (a và a gọi là hai mệnh đề đối ngẫu nhau)

B5 : Xây dựng một mệnh đề mới bằng cách tuyển một cặp mệnh đề trong danh sách mệnh đề ở bước 2. Nếu mệnh đề mới có các biến mệnh đề đối ngẫu nhau thì các biến đó được loại bỏ.

Ví dụ : &#p   q   r   s   q

Hai mệnh đề   q, q là đối ngẫu nên sẽ được loại bỏ

p   r   s

B6 : Thay thế hai mệnh đề vừa tuyển trong danh sách mệnh đề bằng mệnh đề mới.

Ví dụ :

{ p   q ,   r   s   q , w   r, s   q }

{ p r s , w r, s q }

B7 : Nếu không xây dựng được thêm một mệnh đề mới nào và trong danh sách mệnh đề không có 2 mệnh đề nào đối ngẫu nhau thì vấn đề không được chứng minh.

Ví dụ : Chứng minh rằng

p q, q r, r s, u s p, u

B3: { p q, q r, r s, u s, p, u }

B4 : Có tất cả 6 mệnh đề nhưng chưa có mệnh đề nào đối ngẫu nhau.

B5 : tuyển một cặp mệnh đề (chọn hai mệnh đề có biến đối ngẫu). Chọn hai mệnh đề đầu :

p q q r p r

Danh sách mệnh đề thành :

{ p r , r s, u s, p, u }

Vẫn chưa có mệnh đề đối ngẫu.

Tuyển hai cặp mệnh đề đầu tiên

p r r s p s

Danh sách mệnh đề thành { p s, u s, p, u }

Vẫn chưa có hai mệnh đề đối ngẫu

Tuyển hai cặp mệnh đề đầu tiên

p s u s p u

Danh sách mệnh đề thành : { p u, p, u }

Vẫn chưa có hai mệnh đề đối ngẫu

Tuyển hai cặp mệnh đề :

p u u p

Danh sách mệnh đề trở thành : { p, p }

Có hai mệnh đề đối ngẫu nên biểu thức ban đầu đã được chứng minh.

VIII. BIỂU DIỄN TRI THỨC SỬ DỤNG LUẬT DẪN XUẤT (LUẬT SINH)

VIII.1. Khái niệm

Phương pháp biểu diễn tri thức bằng luật sinh được phát minh bởi Newell và Simon trong lúc hai ông đang cố gắng xây dựng một hệ giải bài toán tổng quát. Đây là một kiểu biểu diễn tri thức có cấu trúc. Ý tưởng cơ bản là tri thức có thể được cấu trúc bằng một cặp điều kiện – hành động : "NẾU điều kiện xảy ra THÌ hành động sẽ được thi hành". Chẳng hạn : NẾU đèn giao thông là đỏ THÌ bạn không được đi thẳng, NẾU máy tính đã mở mà không khởi động được THÌ kiểm tra nguồn điện, …

Ngày nay, các luật sinh đã trở nên phổ biến và được áp dụng rộng rãi trong nhiều hệ thống trí tuệ nhân tạo khác nhau. Luật sinh có thể là một công cụ mô tả để giải quyết các vấn đề thực tế thay cho các kiểu phân tích vấn đề truyền thống. Trong trường hợp này, các luật được dùng như là những chỉ dẫn (tuy có thể không hoàn chỉnh) nhưng rất hữu ích để trợ giúp cho các quyết định trong quá trình tìm kiếm, từ đó làm giảm không gian tìm kiếm. Một ví dụ khác là luật sinh có thể được dùng để bắt chước hành vi của những chuyên gia. Theo cách này, luật sinh không chỉ đơn thuần là một kiểu biểu diễn tri thức trong máy tính mà là một kiểu biễu diễn các hành vi của con người.

Một cách tổng quát luật sinh có dạng như sau :

P1   P2   ...   Pn   Q

Tùy vào các vấn đề đang quan tâm mà luật sinh có những ngữ nghĩa hay cấu tạo khác nhau :

Trong logic vị từ : P1, P2, ..., Pn, Q là những biểu thức logic.

Trong ngôn ngữ lập trình, mỗi một luật sinh là một câu lệnh.

IF (P1 AND P2 AND .. AND Pn) THEN Q.

Trong lý thuyết hiểu ngôn ngữ tự nhiên, mỗi luật sinh là một phép dịch :

ONE   một.

TWO   hai.

JANUARY   tháng một

Để biễu diễn một tập luật sinh, người ta thường phải chỉ rõ hai thành phần chính sau :

(1) Tập các sự kiện F(Facts)

F = { f1, f2, ... fn }

(2) Tập các quy tắc R (Rules) áp dụng trên các sự kiện dạng như sau :

f1 ^ f2 ^ ... ^ fi   q

Trong đó, các fi , q đều thuộc F

Ví dụ : Cho 1 cơ sở tri thức được xác định như sau :

Các sự kiện : A, B, C, D, E, F, G, H, K

Tập các quy tắc hay luật sinh (rule)

R1 : A   E

R2 : B   D

R3 : H   A

R4 : E   G   C

R5 : E   K   B

R6 : D   E   K   C

R7 : G   K   F   A

VIII.2. Cơ chế suy luận trên các luật sinh



Suy diễn tiến : là quá trình suy luận xuất phát từ một số sự kiện ban đầu, xác định các sự kiện có thể được "sinh" ra từ sự kiện này.

Sự kiện ban đầu : H, K

R3 : H   A {A, H. K }

R1 : A   E { A, E, H, H }

R5 : E   K   B { A, B, E, H, K }

R2 : B   D { A, B, D, E, H, K }

R6 : D   E   K   C { A, B, C, D, E, H, K }



Suy diễn lùi : là quá trình suy luận ngược xuất phát từ một số sự kiện ban đầu, ta tìm kiếm các sự kiện đã "sinh" ra sự kiện này. Một ví dụ thường gặp trong thực tế là xuất phát từ các tình trạng của máy tính, chẩn đoán xem máy tính đã bị hỏng hóc ở đâu.

Ví dụ :

Tập các sự kiện :

- Ổ cứng là "hỏng" hay "hoạt động bình thường"
- Hỏng màn hình.
- Lỏng cáp màn hình.
- Tình trạng đèn ổ cứng là "tắt" hoặc "sáng"
- Có âm thanh đọc ổ cứng.
- Tình trạng đèn màn hình "xanh" hoặc "chớp đỏ"
- Không sử dụng được máy tính.
- Điện vào máy tính "có" hay "không"

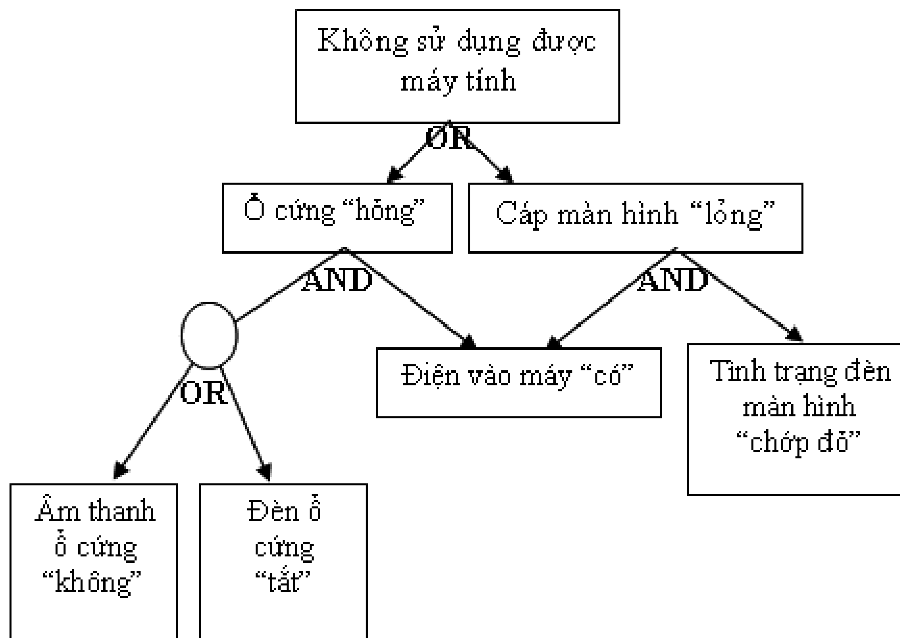
Tập các luật :

R1. Nếu ((ổ cứng "hỏng") hoặc (cáp màn hình "lỏng")) thì không sử dụng được máy tính.

R2. Nếu (điện vào máy là "có") và ((âm thanh đọc ổ cứng là "không") hoặc tình trạng đèn ổ cứng là "tắt")) thì (ổ cứng "hỏng").

R3. Nếu (điện vào máy là "có") và (tình trạng đèn màn hình là "chớp đỏ") thì (cáp màn hình "lỏng").

Để xác định được các nguyên nhân gây ra sự kiện "không sử dụng được máy tính", ta phải xây dựng một cấu trúc đồ thị gọi là đồ thị AND/OR như sau :



Như vậy là để xác định được nguyên nhân gây ra hỏng hóc là do ổ cứng hỏng hay cáp màn hình lỏng, hệ thống phải lần lượt đi vào các nhánh để kiểm tra các điều kiện như điện vào máy "có", âm thanh ổ cứng "không"…Tại một bước, nếu giá trị cần xác định không thể được suy ra từ bất kỳ một luật nào, hệ thống sẽ yêu cầu người dùng trực tiếp nhập vào. Chẳng hạn như để biết máy tính có điện không, hệ thống sẽ hiện ra màn hình câu hỏi "Bạn kiểm tra xem có điện vào máy tính không (kiểm tra đèn nguồn)? (C/K)". Để thực hiện được cơ chế suy luận lùi, người ta thường sử dụng ngăn xếp (để ghi nhận lại những nhánh chưa kiểm tra).

VIII.3. Vấn đề tối ưu luật

Tập các luật trong một cơ sở tri thức rất có khả năng thừa, trùng lắp hoặc mâu thuẫn. Dĩ nhiên là hệ thống có thể đổ lỗi cho người dùng về việc đưa vào hệ thống những tri thức như vậy. Tuy việc tối ưu một cơ sở tri thức về mặt tổng quát là một thao tác khó (vì giữa các tri thức thường có quan hệ không tường minh), nhưng trong giới hạn cơ sở tri thức dưới dạng luật, ta vẫn có một số thuật toán đơn giản để loại bỏ các vấn đề này.

VIII.3.1. Rút gọn bên phải

Luật sau hiển nhiên đúng :

A B A (1)

Do đó luật

A B A C

Là hoàn toàn tương đương với

A B C

Quy tắc rút gọn : Có thể loại bỏ những sự kiện bên vế phải nếu những sự kiện đó đã xuất hiện bên vế trái. Nếu sau khi rút gọn mà vế phải trở thành rỗng thì luật đó là luật hiển nhiên. Ta có thể loại bỏ các luật hiển nhiên ra khỏi tri thức.

VIII.3.2. Rút gọn bên trái

Xét các luật :

(L1) A, B C (L2) A X (L3) X C

Rõ ràng là luật A, B C có thể được thay thế bằng luật A C mà không làm ảnh hưởng đến các kết luận trong mọi trường hợp. Ta nói rằng sự kiện B trong luật (1) là dư thừa và có thể được loại bỏ khỏi luật dẫn trên.

VIII.3.3. Phân rã và kết hợp luật

Luật A B C

Tương đương với hai luật

A C

B C

Với quy tắc này, ta có thể loại bỏ hoàn toàn các luật có phép nối HOẶC. Các luật có phép nối này thường làm cho thao tác xử lý trở nên phức tạp.

VIII.3.4. Luật thừa

Một luật dẫn A B được gọi là thừa nếu có thể suy ra luật này từ những luật còn lại.

Ví dụ : trong tập các luật gồm {A B, B C, A C} thì luật thứ 3 là luật thừa vì nó có thể được suy ra từ 2 luật còn lại.

VIII.3.5. Thuật toán tối ưu tập luật dẫn

Thuật toán này sẽ tối ưu hóa tập luật đã cho bằng cách loại đi các luật có phép nối HOẶC, các luật hiển nhiên hoặc các luật thừa.

Thuật toán bao gồm các bước chính

B1 : Rút gọn vế phải

Với mỗi luật r trong R

Với mỗi sự kiện A VếPhải(r)

Nếu A VếTrái(r) thì Loại A ra khỏi vế phải của R.

Nếu VếPhải(r) rỗng thì loại bỏ r ra khỏi hệ luật dẫn : R = R – {r}

B2 : Phân rã các luật

Với mỗi luật r : X1 X2 … Xn Y trong R

Với mỗi i từ 1 đến n R := R + { Xi Y }

R := R – {r}

B3 : Loại bỏ luật thừa

Với mỗi luật r thuộc R

Nếu VếPhải(r)   BaoĐóng(VếTrái(r), R-{r}) thì R := R – {r}

B4 : Rút gọn vế trái

Với mỗi luật dẫn r : X : A1   A2, …, An   Y thuộc R

Với mỗi sự kiện Ai thuộc r

Gọi luật r1 : X – Ai   Y

S = (R – {r})   {r1}

Nếu BaoĐóng(X – Ai , S)   BaoĐóng(X, R) thì loại sự kiện A ra khỏi X

VIII.4. Ưu điểm và nhược điểm của biểu diễn tri thức bằng luật



Ưu điểm

Biểu diễn tri thức bằng luật đặc biệt hữu hiệu trong những tình huống hệ thống cần đưa ra những hành động dựa vào những sự kiện có thể quan sát được. Nó có những ưu điểm chính yếu sau đây :



Các luật rất dễ hiểu nên có thể dễ dàng dùng để trao đổi với người dùng (vì nó là một trong những dạng tự nhiên của ngôn ngữ).



Có thể dễ dàng xây dựng được cơ chế suy luận và giải thích từ các luật.



Việc hiệu chỉnh và bảo trì hệ thống là tương đối dễ dàng.



Có thể cải tiến dễ dàng để tích hợp các luật mờ.



Các luật thường ít phụ thuộc vào nhau.



Nhược điểm



Các tri thức phức tạp đôi lúc đòi hỏi quá nhiều (hàng ngàn) luật sinh. Điều này sẽ làm nảy sinh nhiều vấn đề liên quan đến tốc độ lẫn quản trị hệ thống.



Thống kê cho thấy, người xây dựng hệ thống trí tuệ nhân tạo thích sử dụng luật sinh hơn tất cả phương pháp khác (dễ hiểu, dễ cài đặt) nên họ thường tìm mọi cách để biểu diễn tri thức bằng luật sinh cho dù có phương pháp khác thích hợp hơn! Đây là nhược điểm mang tính chủ quan của con người.



Cơ sở tri thức luật sinh lớn sẽ làm giới hạn khả năng tìm kiếm của chương trình điều khiển. Nhiều hệ thống gặp khó khăn trong việc đánh giá các hệ dựa trên luật sinh cũng như gặp khó khăn khi suy luận trên luật sinh.

X. BIỄU DIỄN TRI THỨC SỬ DỤNG MẠNG NGỮ NGHĨA

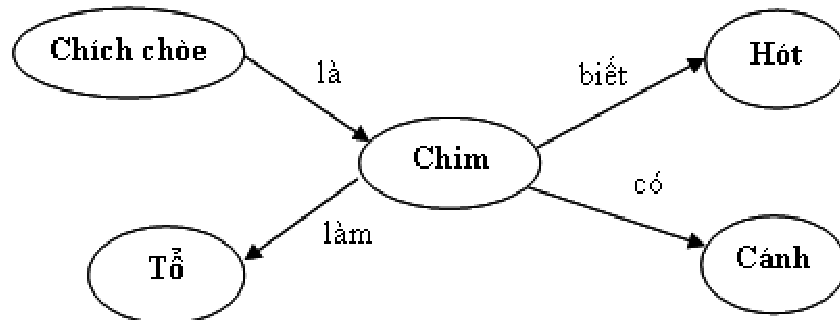
X.1. Khái niệm

Mạng ngữ nghĩa là một phương pháp biểu diễn tri thức đầu tiên và cũng là phương pháp dễ hiểu nhất đối với chúng ta. Phương pháp này sẽ biểu diễn tri thức dưới dạng một đồ thị, trong đó đỉnh là các đối tượng (khái niệm) còn các cung cho biết mối quan hệ giữa các đối tượng (khái niệm) này.

Chẳng hạn : giữa các khái niệm chích chòe, chim, hót, cánh, tổ có một số mối quan hệ như sau :

- Chích chòe là một loài chim.
- Chim biết hót
- Chim có cánh
- Chim sống trong tổ

Các mối quan hệ này sẽ được biểu diễn trực quan bằng một đồ thị như sau :



Do mạng ngữ nghĩa là một loại đồ thị cho nên nó thừa hưởng được tất cả những mặt mạnh của công cụ này. Nghĩa là ta có thể dùng những thuật toán của đồ thị trên mạng ngữ nghĩa như thuật toán tìm liên thông, tìm đường đi ngắn nhất,… để thực hiện các cơ chế suy luận. Điểm đặc biệt của mạng ngữ nghĩa so với đồ thị thông thường chính là việc gán một ý nghĩa (có, làm, là, biết, ...) cho các cung. Trong đồ thị tiêu chuẩn, việc có một cung nối giữa hai đỉnh chỉ cho biết có sự liên hệ giữa hai đỉnh đó và tất cả các cung trong đồ thị đều biểu diễn cho cùng một loại liên hệ. Trong mạng ngữ nghĩa, cung nối giữa hai đỉnh còn cho biết giữa hai khái niệm tương ứng có sự liên hệ như thế nào. Việc gán ngữ nghĩa vào các cung của đồ thị đã giúp giảm bớt được số lượng đồ thị cần phải

dùng để biễu diễn các mối liên hệ giữa các khái niệm. Chẳng hạn như trong ví dụ trên, nếu sử dụng đồ thị thông thường, ta phải dùng đến 4 loại đồ thị cho 4 mối liên hệ : một đồ thị để biểu diễn mối liên hệ "là", một đồ thị cho mối liên hệ "làm", một cho "biết" và một cho "có".

Một điểm khá thú vị của mạng ngữ nghĩa là tính kế thừa. Bởi vì ngay từ trong khái niệm, mạng ngữ nghĩa đã hàm ý sự phân cấp (như các mối liên hệ "là") nên có nhiều đỉnh trong mạng mặc nhiên sẽ có những thuộc tính của những đỉnh khác. Chẳng hạn theo mạng ngữ nghĩa ở trên, ta có thể dễ dàng trả lời "có" cho câu hỏi : "Chích chòe có làm tổ không?". Ta có thể khẳng định được điều này vì đỉnh "chích chòe" có liên kết "là" với đỉnh "chim" và đỉnh "chim" lại liên kết "biết" với đỉnh "làm tổ" nên suy ra đỉnh "chích chòe" cũng có liên kết loại "biết" với đỉnh "làm tổ". (Nếu để ý, bạn sẽ nhận ra được kiểu "suy luận" mà ta vừa thực hiện bắt nguồn từ thuật toán "loang" hay "tìm liên thông" trên đồ thị!). Chính đặc tính kế thừa của mạng ngữ nghĩa đã cho phép ta có thể thực hiện được rất nhiều phép suy diễn từ những thông tin sẵn có trên mạng.

Tuy mạng ngữ nghĩa là một kiểu biểu diễn trực quan đối với con người nhưng khi đưa vào máy tính, các đối tượng và mối liên hệ giữa chúng thường được biểu diễn dưới dạng những phát biểu động từ (như vị từ). Hơn nữa, các thao tác tìm kiếm trên mạng ngữ nghĩa thường khó khăn (đặc biệt đối với những mạng có kích thước lớn). Do đó, mô hình mạng ngữ nghĩa được dùng chủ yếu để phân tích vấn đề. Sau đó, nó sẽ được chuyển đổi sang dạng luật hoặc frame để thi hành hoặc mạng ngữ nghĩa sẽ được dùng kết hợp với một số phương pháp biểu diễn khác.

X.2. Ưu điểm và nhược điểm của mạng ngữ nghĩa



Ưu điểm



Mạng ngữ nghĩa rất linh động, ta có thể dễ dàng thêm vào mạng các đỉnh hoặc cung mới để bổ sung các tri thức cần thiết.



Mạng ngữ nghĩa có tính trực quan cao nên rất dễ hiểu.



Mạng ngữ nghĩa cho phép các đỉnh có thể thừa kế các tính chất từ các đỉnh khác thông qua các cung loại "là", từ đó, có thể tạo ra các liên kết "ngầm" giữa những đỉnh không có liên kết trực tiếp với nhau.



Mạng ngữ nghĩa hoạt động khá tự nhiên theo cách thức con người ghi nhận thông tin.



Nhược điểm



Cho đến nay, vẫn chưa có một chuẩn nào quy định các giới hạn cho các đỉnh và cung của mạng. Nghĩa là bạn có thể gán ghép bất kỳ khái niệm nào cho đỉnh hoặc cung!



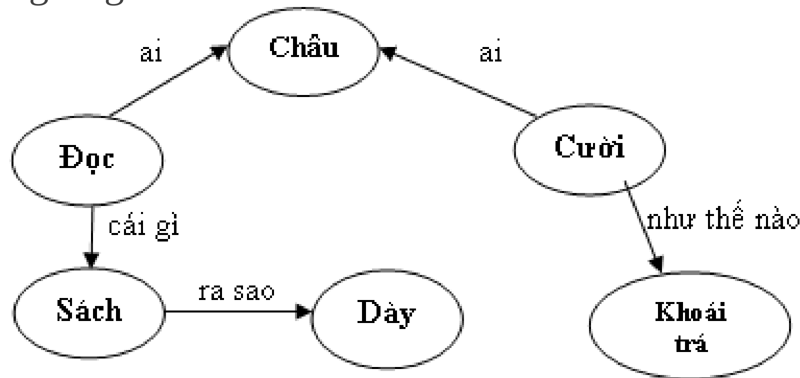
Tính thừa kế (vốn là một ưu điểm) trên mạng sẽ có thể dẫn đến nguy cơ mâu thuẫn trong tri thức. Chẳng hạn, nếu bổ sung thêm nút "Gà" vào mạng như hình sau thì ta có thể kết luận rằng "Gà" biết "bay"!. Sở dĩ có điều này là vì có sự không rõ ràng trong ngữ nghĩa gán cho một nút của mạng. Bạn đọc có thể phản đối quan điểm vì cho rằng, việc sinh ra mâu thuẫn là do ta thiết kế mạng dở chứ không phải do khuyết điểm của mạng!. Tuy nhiên, xin lưu ý rằng, tính thừa kế sinh ra rất nhiều mối liên "ngầm" nên khả năng nảy sinh ra một mối liên hệ không hợp lệ là rất lớn!

Hầu như không thể biểu diễn các tri thức dạng thủ tục bằng mạng ngữ nghĩa vì các khái niệm về thời gian và trình tự không được thể hiện tường minh trên mạng ngữ nghĩa.

X.3. Một ví dụ tiêu biểu

Dù là một phương pháp tương đối cũ và có những yếu điểm nhưng mạng ngữ nghĩavẫn có những ứng dụng vô cùng độc đáo. Hai loại ứng dụng tiêu biểu của mạng ngữ nghĩa là ứng dụng xử lý ngôn ngữ tự nhiên và ứng dụng giải bài toán tự động.

Ví dụ 1 : Trong ứng dụng xử lý ngôn ngữ tự nhiên, mạng ngữ nghĩa có thể giúp máy tính phân tích được cấu trúc của câu để từ đó có thể phần nào "hiểu" được ý nghĩa của câu. Chẳng hạn, câu "Châu đang đọc một cuốn sách dày và cười khoái trá" có thể được biểu diễn bằng một mạng ngữ nghĩa như sau :



Ví dụ 2 : Giải bài toán tam giác tổng quát

Chúng ta sẽ không đi sâu vào ví dụ 1 vì đây là một vấn đề quá phức tạp để có thể trình bày trong cuốn sách này. Trong ví dụ này, chúng ta sẽ khảo sát một vấn đề đơn giản hơn nhưng cũng không kém phần độc đáo. Khi mới học lập trình, bạn thường được giáo viên cho những bài tập nhập môn đại loại như "Cho 3 cạnh của tam giác, tính chiều dài các đường cao", "Cho góc a, b và cạnh AC. Tính chiều dài trung tuyến", ... Với mỗi bài tập này, việc bạn cần làm là lấy giấy bút ra tìm cách tính, sau khi đã xác định các bước tính toán, bạn chuyển nó thành chương trình. Nếu có 10 bài, bạn phải làm lại việc tính toán rồi lập trình 10 lần. Nếu có 100 bài, bạn phải làm 100 lần. Và tin buồn cho bạn là số lượng bài toán thuộc loại này là rất nhiều! Bởi vì một tam giác có tất cả 22 yếu tố khác nhau!. Không lẽ mỗi lần gặp một bài toán mới, bạn đều phải lập trình lại? Liệu có một chương trình tổng quát có thể tự động giải được

tất cả (vài ngàn!) những bài toán tam giác thuộc loại này không? Câu trả lời là CÓ ! Và ngạc nhiên hơn nữa, chương trình này lại khá đơn giản. Bài toán này sẽ được giải bằng mạng ngữ nghĩa.

Có 22 yếu tố liên quan đến cạnh và góc của tam giác. Để xác định một tam giác hay để xây dựng một 1 tam giác ta cần có 3 yếu tố trong đó phải có yếu tố cạnh. Như vậy có khoảng C322 -1 (khoảng vài ngàn) cách để xây dựng hay xác định một tam giác. Theo thống kê, có khoảng 200 công thức liên quan đến cạnh và góc 1 tam giác.

Để giải bài toán này bằng công cụ mạng ngữ nghĩa, ta phải sử dụng khoảng 200 đỉnh để chứa công thức và khoảng 22 đỉnh để chứa các yếu tố của tam giác. Mạng ngữ nghĩa cho bài toán này có cấu trúc như sau :

Đỉnh của đồ thị bao gồm hai loại :

Đỉnh chứa công thức (ký hiệu bằng hình chữ nhật)

Đỉnh chứa yếu tố của tam giác (ký hiệu bằng hình tròn)

Cung : chỉ nối từ đỉnh hình tròn đến đỉnh hình chữ nhật cho biết yếu tố tam giác xuất hiện trong công thức nào (không có trường hợp cung nối giữa hai đỉnh hình tròn hoặc cung nối giữa hai đỉnh hình chữ nhật).

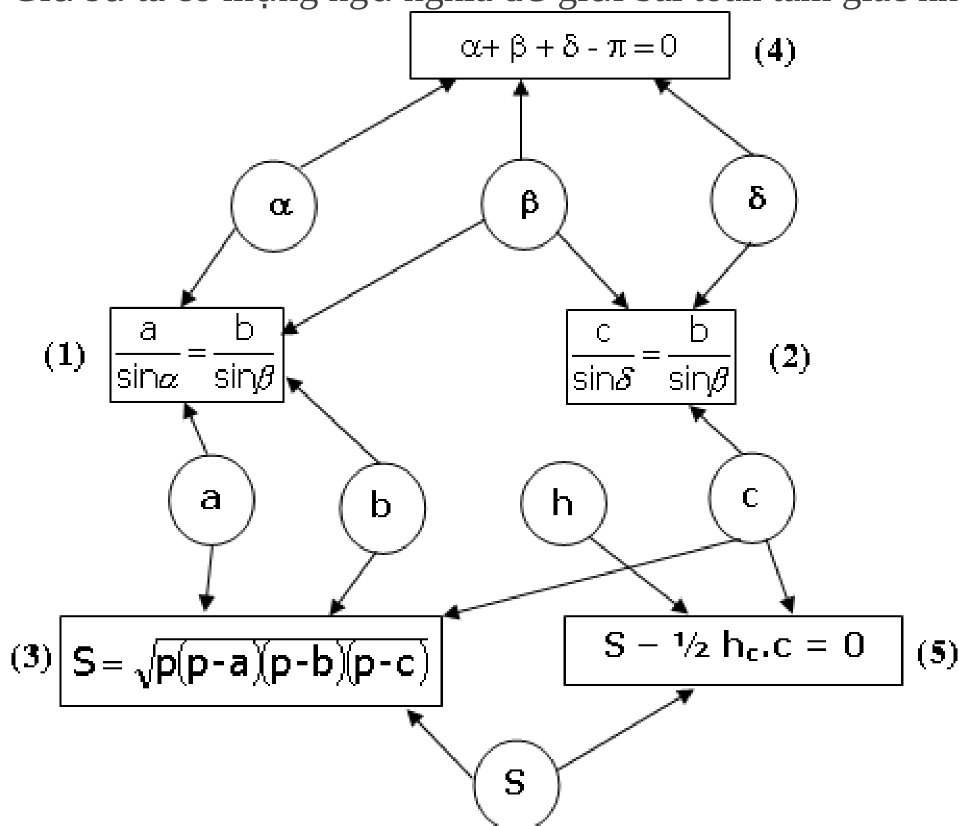
* Lưu ý : trong một công thức liên hệ giữa n yếu tố của tam giác, ta giả định rằng nếu đã biết giá trị của n-1 yếu tố thì sẽ tính được giá trị của yếu tố còn lại. Chẳng hạn như trong công thức tổng 3 góc của tam giác bằng 1800 thì khi biết được hai góc, ta sẽ tính được góc còn lại.

Cơ chế suy diễn thực hiện theo thuật toán "loang" đơn giản sau :

B1 : Kích hoạt những đỉnh hình tròn đã cho ban đầu (những yếu tố đã có giá trị)B2 : Lặp lại bước sau cho đến khi kích hoạt được tất cả những đỉnh ứng với những yếu tố cần tính hoặc không thể kích hoạt được bất kỳ đỉnh nào nữa. Nếu một đỉnh hình chữ nhật có cung nối với n đỉnh hình tròn mà n-1 đỉnh hình tròn đã được kích hoạt thì kích hoạt đỉnh hình tròn

còn lại (và tính giá trị đỉnh còn lại này thông qua công thức ở đỉnh hình chữ nhật).

Giả sử ta có mạng ngữ nghĩa để giải bài toán tam giác như hình sau



Ví dụ : "Cho hai góc và chiều dài cạnh a của tam giác. Tính chiều dài đường cao hC". Với mạng ngữ nghĩa đã cho trong hình trên. Các bước thi hành của thuật toán như sau :

Bắt đầu : đỉnh a của đồ thị được kích hoạt.

Công thức (1) được kích hoạt (vì a được kích hoạt). Từ công thức (1) tính được cạnh b. Đỉnh b được kích hoạt.

Công thức (4) được kích hoạt (vì). Từ công thức (4) tính được góc

Công thức (2) được kích hoạt (vì 3 đỉnh b được kích hoạt). Từ công thức (2) tính được cạnh c. Đỉnh c được kích hoạt.

Công thức (3) được kích hoạt (vì 3 đỉnh a, b, c được kích hoạt) . Từ công thức (3) tính được diện tích S. Đỉnh S được kích hoạt.

Công thức (5) được kích hoạt (vì 2 đỉnh S, c được kích hoạt). Từ công thức (5) tính được hC. Đỉnh hC được kích hoạt.

Giá trị hC đã được tính. Thuật toán kết thúc.

Về mặt chương trình, ta có thể cài đặt mạng ngữ nghĩa giải bài toán tam giác bằng một mảng hai chiều A trong đó :

Cột : ứng với công thức. Mỗi cột ứng với một công thức tam giác khác nhau (đỉnh hình chữ nhật).

Dòng : ứng với yếu tố tam giác. Mỗi dòng ứng với một yếu tố tam giác khác nhau (đỉnh hình tròn).

Phần tử A[i, j] = -1 nghĩa là trong công thức ứng với cột j có yếu tố tam giác ứng với cột i. Ngược lại A[i,j] = 0.

Để thực hiện thao tác "kích hoạt" một đỉnh hình tròn, ta đặt giá trị của toàn dòng ứng với yếu tố tam giác bằng 1.

Để kiểm tra xem một công thức đã có đủ n-1 yếu tố hay chưa (nghĩa là kiểm tra điều kiện "đỉnh hình chữ nhật có cung nối với n đỉnh hình tròn mà n-1 đỉnh hình tròn đã được kích hoạt"), ta chỉ việc lấy hiệu giữa tổng số ô có giá trị bằng 1 và tổng số ô có giá trị -1 trên cột ứng với công thức cần kiểm tra. Nếu kết quả bằng n, thì công thức đã có đủ n-1 yếu tố.

Trở lại mạng ngữ nghĩa đã cho. Quá trình thi hành kích hoạt được diễn ra như sau :

Mảng biểu diễn mạng ngữ nghĩa ban đầu

	(1)	(2)	(3)	(4)	(5)
	-1	0	0	-1	0
	-1	-1	0	-1	0
	0	-1	0	-1	0
a	-1	0	-1	0	0
b	-1	-1	-1	0	0
c	0	-1	-1	0	-1
S	0	0	-1	0	-1
hC	0	0	0	0	-1

Khởi đầu : đỉnh , a của đồ thị được kích hoạt.

	(1)	(2)	(3)	(4)	(5)
	1	0	0	1	0
	1	1	0	1	0
	0	-1	0	-1	0
a	1	0	1	1	0
b	-1	-1	-1	0	0

c	0	-1	-1	0	-1
S	0	0	-1	0	-1
hC	0	0	0	0	-1

Trên cột (1), hiệu (1+1+1 – (-1)) = 4 nên dòng b sẽ được kích hoạt.

	(1)	(2)	(3)	(4)	(5)
	1	0	0	1	0
	1	1	0	1	0
	0	-1	0	-1	0
a	1	0	1	1	0
b	1	1	1	0	0
c	0	-1	-1	0	-1
S	0	0	-1	0	-1
hC	0	0	0	0	-1

Trên cột (4), hiệu (1+1+1 – (-1)) = 4 nên dòng   sẽ được kích hoạt.

	(1)	(2)	(3)	(4)	(5)
	1	0	0	1	0
	1	1	0	1	0
	0	1	0	1	0
a	1	0	1	1	0
b	1	1	1	0	0
c	0	-1	-1	0	-1
S	0	0	-1	0	-1
hC	0	0	0	0	-1

Trên cột (2), hiệu (1+1+1 – (1)) = 4 nên dòng c được kích hoạt.

	(1)	(2)	(3)	(4)	(5)
	1	0	0	1	0
	1	1	0	1	0
	0	1	0	1	0
A	1	0	1	1	0
B	1	1	1	0	0

C	0	1	1	0	1
S	0	0	-1	0	-1
hC	0	0	0	0	-1

Trên cột (3), hiệu (1+1+1 – (-1)) = 4 nên dòng S được kích hoạt.

	(1)	(2)	(3)	(4)	(5)
	1	0	0	1	0
	1	1	0	1	0
	0	1	0	1	0
a	1	0	1	1	0
b	1	1	1	0	0
c	0	1	1	0	1
S	0	0	1	0	1
hC	0	0	0	0	-1

Trên cột (5), hiệu (1+1 – (1)) = 3 nên dòng hC được kích hoạt.

Khả năng của hệ thống này không chỉ dừng lại ở việc tính ra giá trị các yếu tố cần thiết, với một chút sửa đổi, chương trình này còn có thể đưa ra cách giải hình thức của bài toán và thậm chí còn có thể chọn được

cách giải hình thức tối ưu (tối ưu hiểu theo nghĩa là cách giải sử dụng những công thức đơn giản nhất). Sở dĩ có thể nói như vậy vì cách suy luận của ta trong bài toán này là tìm kiếm theo chiều rộng. Do đó, khi đạt đến kết quả, ta có thể có rất nhiều cách khác nhau. Để có thể chọn được giải pháp tối ưu, bạn cần phải định nghĩa được độ "phức tạp" của một công thức. Một trong những tiêu chuẩn thường được dùng là số lượng phép nhân, chia, cộng, trừ, rút căn, tính sin, cos, ... được áp dụng trong công thức. Các phép tính sin, cos và rút căn có độ phức tạp cao nhất, kế đến là nhân chia và cuối cùng là cộng trừ. Cuối cùng bạn có thể cải tiến lại phương pháp suy luận bằng cách vận dụng thuật toán A với ước lượng h=0 để có thể chọn ra được "đường đi" tối ưu. Ta chọn ước lượng h=0 vì hai lý do sau (1) không gian bài toán nhỏ nên ta không cần phải giới hạn độ rộng tìm kiếm (2) xây dựng một ước lượng như vậy là tương đối khó khăn, đặc biệt là làm sao để hệ thống không đánh giá quá cao h.

XI. BIỂU DIỄN TRI THỨC BẰNG FRAME

XI.1. Khái niệm

Frame là một cấu trúc dữ liệu chứa đựng tất cả những tri thức liên quan đến một đối tượng cụ thể nào đó. Frames có liên hệ chặt chẽ đến khái niệm hướng đối tượng (thực ra frame là nguồn gốc của lập trình hướng đối tượng). Ngược lại với các phương pháp biểu diễn tri thức đã được đề cập đến, frame "đóng gói" toàn bộ một đối tượng, tình huống hoặc cả một vấn đề phức tạp thành một thực thể duy nhất có cấu trúc. Một frame bao hàm trong nó một khối lượng tương đối lớn tri thức về một đối tượng, sự kiện, vị trí, tình huống hoặc những yếu tố khác. Do đó, frame có thể giúp ta mô tả khá chi tiết một đối tượng.

Dưới một khía cạnh nào đó, người ta có thể xem phương pháp biểu diễn tri thức bằng frame chính là nguồn gốc của ngôn ngữ lập trình hướng đối tượng. Ý tưởng của phương pháp này là "thay vì bắt người dùng sử dụng các công cụ phụ như dao mở để đồ hộp, ngày nay các hãng sản xuất đồ hộp thường gắn kèm các nắp mở đồ hộp ngay bên trên vỏ lon. Như vậy, người dùng sẽ không bao giờ phải lo lắng đến việc tìm một thiết bị để

mở đồ hộp nữa!". Cũng vậy, ý tưởng chính của frame (hay của phương pháp lập trình hướng đối tượng) là khi biểu diễn một tri thức, ta sẽ "gắn kèm" những thao tác thường gặp trên tri thức này. Chẳng hạn như khi mô tả khái niệm về hình chữ nhật, ta sẽ gắn kèm cách tính chu vi, diện tích.

Frame thường được dùng để biểu diễn những tri thức "chuẩn" hoặc những tri thức được xây dựng dựa trên những kinh nghiệm hoặc các đặc điểm đã được hiểu biết cặn kẽ. Bộ não của con người chúng ta vẫn luôn "lưu trữ" rất nhiều các tri thức chung mà khi cần, chúng ta có thể "lấy ra" để vận dụng nó trong những vấn đề cần phải giải quyết. Frame là một công cụ thích hợp để biểu diễn những kiểu tri thức này.

XI.2. Cấu trúc của frame

Mỗi một frame mô tả một đối tượng (object). Một frame bao gồm 2 thành phần cơ bản là slot và facet. Một slot là một thuộc tính đặc tả đối tượng được biểu diễn bởi frame. Ví dụ : trong frame mô tả xe hơi, có hai slot là trọng lượng và loại máy.

Mỗi slot có thể chứa một hoặc nhiều facet. Các facet (đôi lúc được gọi là slot "con") đặc tả một số thông tin hoặc thủ tục liên quan đến thuộc tính được mô tả bởi slot. Facet có nhiều loại khác nhau, sau đây là một số facet thường gặp.



Value (giá trị) : cho biết giá trị của thuộc tính đó (như xanh, đỏ, tím vàng nếu slot là màu xe).



Default (giá trị mặc định) : hệ thống sẽ tự động sử dụng giá trị trong facet này nếu slot là rỗng (nghĩa là chẳng có đặc tả nào!). Chẳng hạn trong frame về xe, xét slot về số lượng bánh. Slot này sẽ có giá trị 4. Nghĩa là, mặc định một chiếc xe hơi sẽ có 4 bánh!



Range (miền giá trị) : (tương tự như kiểu biến), cho biết giá trị slot có thể nhận những loại giá trị gì (như số nguyên, số thực, chữ cái, ...)



If added : mô tả một hành động sẽ được thi hành khi một giá trị trong slot được thêm vào (hoặc được hiệu chỉnh). Thủ tục thường được viết dưới dạng một script.



If needed : được sử dụng khi slot không có giá trị nào. Facet mô tả một hàm để tính ra giá trị của slot.

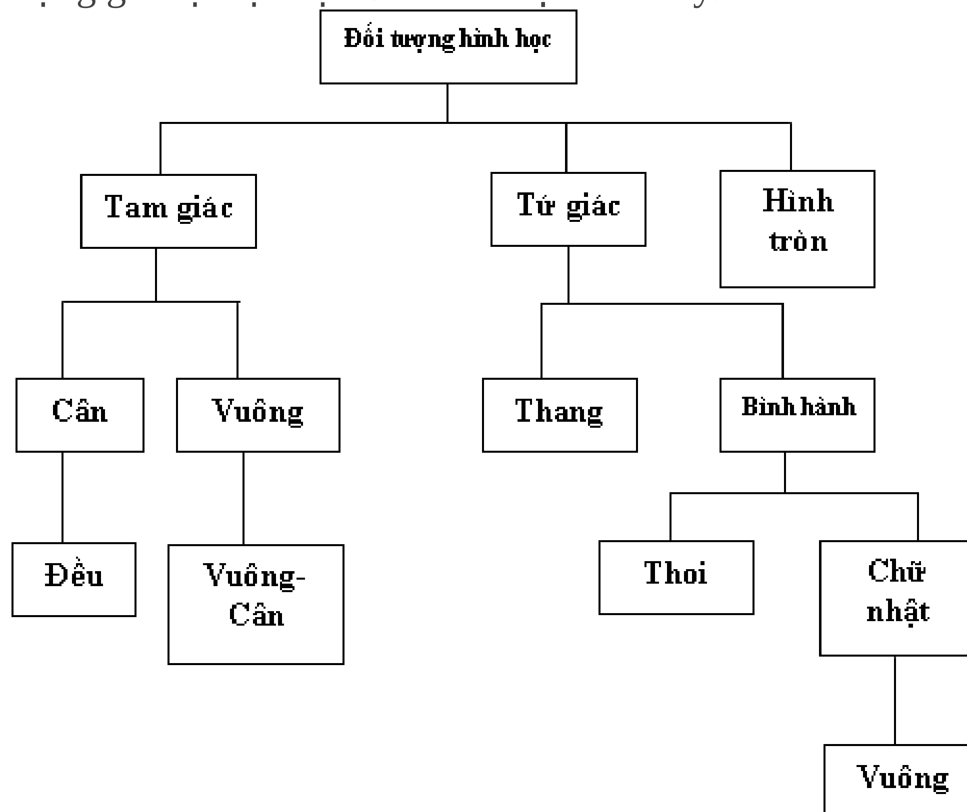
Frame : XE HƠI Thuộc lớp : phương tiện vận chuyển.Tên nhà sản xuất : AudiQuốc gia của nhà sản xuất : ĐứcModel : 5000 TurboLoại xe : SedanTrọng lượng : 3300lbSố lượng cửa : 4 (default)Hộp số : 3 số tự độngSố lượng bánh : 4 (default)Máy (tham chiếu đến frame Máy)Kiểu : In-line, overhead camSố xy-lanh : 5Khả năng tăng tốc 0-60 : 10.4 giây¼ dặm : 17.1 giây, 85 mph.		Frame MÁY Xy-lanh : 3.19 inchTỷ lệ nén : 3.4 incheXăng : TurboChargerMã lực : 140 hp

XI.3. Tính kế thừa

Trong thực tế, một hệ thống trí tuệ nhân tạo thường sử dụng nhiều frame được liên kết với nhau theo một cách nào đó. Một trong những điểm thú vị của frame là tính phân cấp. Đặc tính này cho phép kế thừa các tính chất giữa các frame.

Hình sau đây cho thấy cấu trúc phân cấp của các loại hình hình học cơ bản. Gốc của cây ở trên cùng tương ứng với mức độ trừu tượng cao nhất. Các frame nằm ở dưới cùng (không có frame con nào) gọi là lá. Những frame nằm ở mức thấp hơn có thể thừa kế tất cả những tính chất của những frame cao hơn.

Các frame cha sẽ cung cấp những mô tả tổng quát về thực thể. Frame có cấp càng cao thì mức độ tổng quát càng cao. Thông thường, frame cha sẽ bao gồm các định nghĩa của các thuộc tính. Còn các frame con sẽ chứa đựng giá trị thực sự của các thuộc tính này.



Một ví dụ biểu diễn các đối tượng hình học bằng frame

Các kiểu dữ liệu cơ bản :

Area : numeric; // diện tích

Height : numeric; //chiều cao

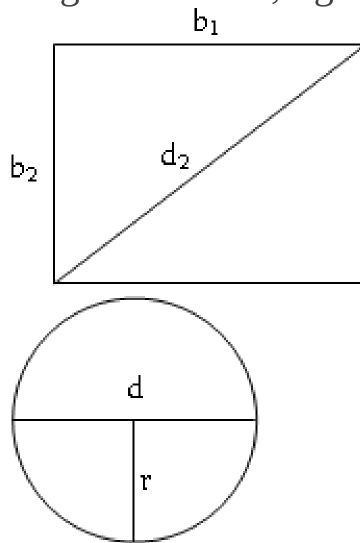
Perimeter : numberic; //chu vi

Side : numeric; //cạnh

Diagonal : numeric; //đường chéo

Radius : numeric; //bán kính

Angle : numeric; //góc



Diameter : numeric; //đường kính

pi : (val:numeric = 3.14159)

Frame : CIRCLE (hình tròn)

r : radius;

s : area;

p : perimeter;

d : diameter;

d = 2   r;

s = pi   r2;

p = 2   pi   r;

Frame RECTANGLE (hình chữ nhật)

b1 : side;

b2 : side;

s : area;

p : perimeter;

s = b1   b2;

p = 2   (b1+b2);

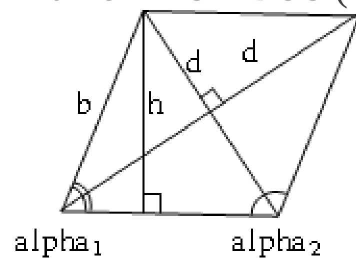
d2 = b12 + b22;

Frame SQUARE (hình vuông)

Là : RECTANGLE

b1 = b2;

Frame RHOMBUS (hình thoi)



b : side;

d1 : diagonal;

d2 : diagonal;

s : area;

p : perimeter;

alpha1 : angle;

alpha2 : angle;

h : height;

cos (alpha2/2)   d1 = h;

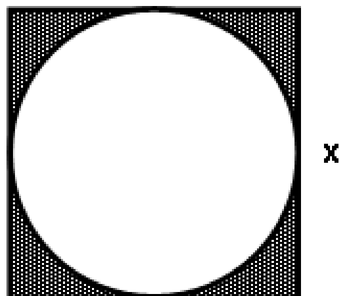
s = d1   d2 / 2;

p = 4   b;

s = b   h;

cos (alpha2/2)/(2 b) = d2;

Chúng ta có thể dễ dàng khai báo các đối tượng hình học khác theo cách này. Sau khi đã biểu diễn các tri thức về các hình hình học cơ bản xong, ta có thể vận dụng nó để giải các bài toán hình học, chẳng hạn bài toán tính diện tích. Ví dụ, cho hình vuông k và vòng tròn nội tiếp c, biết cạnh hình vuông có chiều dài là x, hãy viết chương trình để tính diện tích phần tô đen.



Dễ thấy rằng, diện tích phần tô đen chính là hiệu giữa diện tích hình vuông và diện tích hình tròn nội tiếp. Dĩ nhiên là bạn cũng có thể viết một chương trình bình thường để tính toán, nhưng khi đã "tích hợp" các tri thức về tính diện tích bên trong biểu diễn, chương trình của chúng ta trở nên rất gọn nhẹ. Bạn hãy lưu ý 3 lệnh được in đậm trong ví dụ dưới. Lệnh đầu tiên sẽ "đặc tả" lại giả thiết "hình vuông có cạnh với chiều dài

x", lệnh kế tiếp đặc tả giả thiết "hình tròn nội tiếp", còn lệnh thứ 3 mô tả việc tính diện tích bằng cách lấy diện tích hình vuông trừ cho diện tích hình tròn.

VAR x, s : numeric; k : square; c : circle;

BEGIN

<Nhập x>;

k.b1 := x;

c.d := x;

s := k.s – c.s;

END.

Như vậy, chương trình máy tính của chúng ta đã hoạt động khá giống như việc "mô tả" các giải bài toán bằng ngôn ngữ tự nhiên. Hãy nghĩ xa hơn một tí. Các bài toán hình học thường được mô tả bằng các ngôn từ khá chính xác (chẳng hạn như : cho một tam giác với chiều cao xuất phát từ đỉnh A là 5, chiều dài cạnh đáy là 6, ...). Do đó, về mặt nguyên tác, chúng ta vẫn có thể xây dựng một chương trình để "hiểu" những đề bài này (theo như cách mà chúng ta vừa làm). Sau đó, người dùng có thể hoàn toàn nhờ máy tính giải giúp bài toán cho mình bằng cách mô tả lời giải cho máy tính (chứ không cần phải lập trình). Bạn có cảm giác điều này thật thú vị không? Đây chính là bước đi đầu tiên trong việc tạo ra một chương trình trợ giúp cho việc giải các bài toán hình học trên máy tính với giao tiếp bằng ngôn ngữ tự nhiên!

Để tăng thêm sức mạnh cho hệ thống này, người ta thường cài đặt một mạng ngữ nghĩa ngay bên trong mỗi frame. Chẳng hạn, ta có thể có một frame TRIANGLE, trong đó cài đặt một mạng ngữ nghĩa (giống như ở ví dụ trong phần mạng ngữ nghĩa) để đặc tả mối liên hệ giữa các yếu tố tam giác (thay vì sử dụng các công thức liên hệ đơn giản như ví dụ trên).

XII. BIỂU DIỄN TRI THỨC BẰNG SCRIPT

Script là một cách biểu diễn tri thức tương tự như frame nhưng thay vì đặc tả một đối tượng, nó mô tả một chuỗi các sự kiện. Để mô tả chuỗi sự kiện, script sử dụng một dãy các slot chứa thông tin về các con người, đối tượng và hành động liên quan đến sự kiện đó.

Tuy cấu trúc của các script là rất khác nhau tùy theo bài toán, nhưng nhìn chung một script thường bao gồm các thành phần sau :

- Điều kiện vào (entry condition): mô tả những tình huống hoặc điều kiện cần được thỏa mãn trước khi các sự kiện trong script có thể diễn ra.

- Role (diễn viên): là những con người có liên quan trong script.

- Prop (tác tố): là tất cả những đối tượng được sử dụng trong các chuỗi sự kiện sẽ diễn ra.

- Scene(Tình huống) : là chuỗi sự kiện thực sự diễn ra.

- Result (Kết quả) : trạng thái của các Role sau khi script đã thi hành xong.

- Track (phiên bản) : mô tả một biến thể (hoặc trường hợp đặc biệt) có thể xảy ra trong đoạn script.

Sau đây là một ví dụ tiêu biểu cho script. Ví dụ này là một biến thể của ví dụ nổi tiếng về nhà hàng bán thức ăn nhanh (các nhà hàng bán gà rán mà ta thường gặp trong các siêu thị!) thường được sử dụng để minh họa cách biểu diễn tri thức bằng script trong cách sách nói về trí tuệ nhân tạo. Đi ăn trong một nhà hàng là một tình huống thường gặp trong cuộc sống với những điều kiện vào, diễn viên, tác tố, hoàn cảnh, kết quả khá

"chuẩn". Và qua script ở ví dụ, bạn sẽ thấy phương pháp này có thể được dùng để mô tả chính xác những tình huống diễn ra hàng ngày của những nhà hàng bán thức ăn nhanh. Các tình huống là những đoạn script con trong đoạn script chính để mô tả những tình huống nhỏ trong toàn bộ quá trình. Lưu ý rằng trong đoạn script này có tình huống tùy chọn trong đó mô tả việc khách hàng mua thức ăn về thay vì vào nhà hàng ăn.

Script "nhà hàng"

Phiên bản : Nhà hàng bán thức ăn nhanh.

Diễn viên : Khách hàng

Người phục vụ.

Tác tố : Bàn phục vụ.

Chỗ ngồi.

Khay đựng thức ăn

Thức ăn

Tiền

Các loại gia vị như muối, tương, ớt, tiêu, ...

Điều kiện vào :

Khách hàng đói

Khách hàng có đủ tiền để trả.

Tình huống 1 : Vào nhà hàng

Khách hàng đậu xe vào bãi đậu xe.

Khách hàng bước vào nhà hàng.

Khách hàng xếp hàng trước bàn phục vụ.

Khách hàng đọc thực đơn trên tường và quyết định sẽ kêu món ăn gì.

Tình huống 2: Kêu món ăn.

Khách hàng kêu món ăn với người phục vụ (đang đứng ở quầy phục vụ)

Người phục vụ đặt thức ăn lên khay và đưa hóa đơn tính tiền cho khách.

Khách hàng trả tiền cho người phục vụ.

Tình huống 3: Khách hàng dùng món ăn

Khách hàng lấy thêm các gia vị

Khách hàng cầm khay đến một bàn còn trống.

Khách hàng ăn thức ăn.

Tình huống 3A (tùy chọn) : Khách hàng mua thức ăn đem về

Khách hàng mang thức ăn về nhà.

Tình huống 4 : Ra về

Khách hàng thu dọn bàn

Khách hàng bỏ rác (thức ăn thừa, xương, mảng vụn, ...) vào thùng rác.

Khách hàng ra khỏi nhà hàng.

Khách hàng lái xe đi.

Kết quả :

Khách hàng không còn đói.

Khách hàng còn ít tiền hơn ban đầu.

Khách hàng vui vẻ *

Khách hàng bực mình *

Khách hàng quá no.

* Tùy chọn.

Script rất hữu dụng trong việc dự đoán điều gì sẽ xảy đến trong những tình huống xác định. Thậm chí trong những tình huống chưa diễn ra, script còn cho phép máy tính dự đoán được việc gì sẽ xảy ra và xảy ra đối với ai và vào thời điểm nào. Nếu máy tính kích hoạt một script, người dùng có thể đặt câu hỏi và hệ thống có thể suy ra được những câu trả lời chính xác mà không cần người dùng cung cấp thêm nhiều thông tin (trong một số trường hợp có thể không cần thêm thông tin). Do đó, cũng giống như frame, script là một dạng biểu diễn tri thức tương đối hữu dụng vì nó cho phép ta mô tả chính xác những tình huống "chuẩn" mà con người vẫn thực hiện mỗi ngày hoặc đã nắm bắt chính xác.

Để cài đặt script trong máy tính, bạn phải tìm cách lưu trữ các tri thức dưới dạng hình thức. LISP là ngôn ngữ lập trình phù hợp nhất để làm điều này. Sau khi đã cài đặt xong script, bạn (người dùng) có thể đặt câu hỏi về những con người hoặc điều kiện có liên quan trong script. Hệ thống sau đó sẽ tiến hành thao tác tìm kiếm hoặc thao tác so mẫu để tìm câu trả lời. Chẳng hạn bạn có thể đặt câu hỏi "Khách hàng làm gì trước tiên?". Hệ thống sẽ tìm thấy câu trả lời trong scene 1 và đưa ra đáp án "Đậu xe và bước vào nhà hàng".

XIII. PHỐI HỢP NHIỀU CÁCH BIỂU DIỄN TRI THỨC

Mục tiêu chính biểu diễn tri thức trong máy tính là phục vụ cho việc thu nhận tri thức vào máy tính, truy xuất tri thức và thực hiện các phép suy luận dựa trên những tri thức đã lưu trữ. Do đó, để thỏa mãn được 3 mục tiêu trên, khi chọn phương pháp biểu diễn tri thức, chúng ta phải cân nhắc một số yếu tố cơ bản sau đây :



Tính tự nhiên, đồng bộ và dễ hiểu của biểu diễn tri thức.

- Mức độ trừu tượng của tri thức : tri thức được khai báo cụ thể hay nhúng vào hệ thống dưới dạng các mã thủ tục?

- Tính đơn thể và linh động của cơ sở tri thức (có cho phép dễ dàng bổ sung tri thức, mức độ phụ thuộc giữa các tri thức, ...)

- Tính hiệu quả trong việc truy xuất tri thức và sức mạnh của các phép suy luận (theo kiểu heuristic) .

Bảng sau cho chúng ta một số ưu và khuyết điểm của các phương pháp biểu diễn tri thức đã được trình bày.

P.Pháp	Ưu điểm	Nhược điểm
Luật sinh	Cú pháp đơn giản, dễ hiểu, diễn dịch đơn giản, tính đơn thể cao, linh động (dễ điều chỉnh).	Rất khó theo dõi sự phân cấp, không hiệu quả trong những hệ thống lớn, không thể biểu diễn được mọi loại tri thức, rất yếu trong việc biểu diễn các tri thức dạng mô tả, có cấu trúc.
Mạng ngữ nghĩa	Dễ theo dõi sự phân cấp, sẽ dò theo các mối liên hệ, linh	Ngữ nghĩa gắn liền với mỗi đỉnh có thể

	động	nhập nhằng, khó xử lý các ngoại lệ, khó lập trình.
Frame	Có sức mạnh diễn đạt tốt, dễ cài đặt các thuộc tính cho các slot cũng như các mối liên hệ, dễ dàng tạo ra các thủ tục chuyên biệt hóa, dễ đưa vào các thông tin mặc định và dễ thực hiện các thao tác phát hiện các giá trị bị thiếu sót.	Khó lập trình, khó suy diễn, thiếu phần mềm hỗ trợ.
Logic hình thức	Cơ chế suy luận chính xác (được chứng minh bởi toán học).	Tách rời việc biểu diễn và xử lý, không hiệu quả với lượng dữ liệu lớn, quá chậm khi cơ sở dữ liệu lớn.

Tuy vậy, như chúng ta đã biết, hiện nay vẫn chưa có một kiểu biểu diễn tri thức nào phù hợp với mọi tình huống. Do đó, khi phải làm việc với nhiều nguồn tri thức khác nhau (khác loại, khác tính chất), chúng ta nhiều lúc phải hy sinh tính đồng bộ bằng cách sử dụng cùng lúc nhiều kiểu biểu diễn tri thức, mỗi kiểu biểu diễn ứng với một nhiệm vụ con. Nhưng như vậy, chúng ta lại nảy sinh ra vấn đề "dịch" một tri thức từ kiểu biểu diễn này sang kiểu biểu diễn khác. Tuy thế nhưng một số hệ chương trình trí tuệ gần đây vẫn dùng cùng lúc nhiều kiểu biểu diễn dữ liệu khác nhau.

Một trong những ví dụ kết hợp nhiều kiểu biểu diễn tri thức mà chúng ta đã từng làm quen là kiểu kết hợp giữa frame và mạng ngữ nghĩa trong việc trợ giúp giải bài toán hình học.

Một trong những sự phối hợp tương đối thành công là sự kết hợp giữa luật sinh và frame. Luật sinh không đủ hiệu quả trong nhiều ứng dụng,

đặc biệt là trong các tác vụ định nghĩa, mô tả các đối tượng hoặc những mối liên kết tĩnh giữa các đối tượng. Nhưng những yếu điểm này lại chính là ưu điểm của frame. Ngày nay, đã có rất nhiều hệ thống đã tạo ra một kiểu biểu diễn lai giữa luật sinh và frame có được ưu điểm của hai cách biểu diễn. Sự thành công của các hệ thống nổi tiếng như KEE, Level5 Object và Nexpert Object đã minh chứng cho điều này. Frame cung cấp một ngôn ngữ cấu trúc hiệu quả để đặc tả những đối tượng xuất hiện trong các luật. Frame còn đóng vai trò như một lớp hỗ trợ cho thao tác suy diễn cơ bản trên những đối tượng không cần phải tương tác một cách tường minh trong các luật. Khả năng phân lớp của frame còn có thể được dùng để phân hoạch, tạo chỉ mục và sắp xếp các luật sinh trong hệ thống. Khả năng này rất thích hợp cho người dùng trong việc xây dựng và hiểu các luật, cũng như cũng có thể theo dõi được các luật được sử dụng khi nào và cho mục gì.

Hình sau cho thấy một kiểu kết hợp giữa luật sinh và frame. Sự kết hợp này đã cho phép tạo ra các luật so mẫu nhằm tăng tốc độ tìm kiếm của hệ thống. Kết quả của sự kết hợp này cho phép tạo ra các biểu diễn phức tạp hơn rất nhiều so với việc chỉ dùng frame, thậm chí phức tạp hơn cả việc lập trình trực tiếp bằng ngôn ngữ C++ !!.



* Suy luận không chắc chắn (Hypothetical reasoning) : là kỹ thuật suy luận dựa trên các điều kiện có thể có mâu thuẫn hoặc không chắc chắn.

Ví dụ kết hợp biểu diễn tri thức bằng luật sinh và frame trong bài toán điều chế chất hóa học

Vấn đề : Cho trước một số chất hóa học. Hãy xây dựng chuỗi các phản ứng hóa học để điều chế một số chất hóa học khác.

Đầu tiên, đây là một ứng dụng hết sức tự nhiên của tri thức biểu diễn dưới dạng luật. Lý do là vì bản thân các phản ứng hóa học tiêu chuẩn đều được thể hiện dưới dạng luật. Chẳng hạn ta có các phương trình phản ứng sau :


$Na + Cl_2 \quad NaCl$


$Fe + Cl_2 \quad FeCl_2$


$Cu + Cl_2 \quad CuCl_2$


$Cl_2 + H_2O \quad HCl + HClO$


$MnO_2 + 4HCl \quad MnCl_2 + Cl_2 + H_2O$


$HCl + KMnO_4 \quad KCl + MnCl_2 + H_2O + Cl_2$


$NaCl + H_2O \quad Cl_2 + H_2 + NaOH$

...

Như vậy, nếu xem một chất hóa học là một sự kiện và một phương trình phản ứng như là một luật dẫn thì bài toán điều chế chất hóa học, một cách rất tự nhiên, trở thành bài toán suy luận tiến trong cơ sở tri thức dạng luật dẫn.

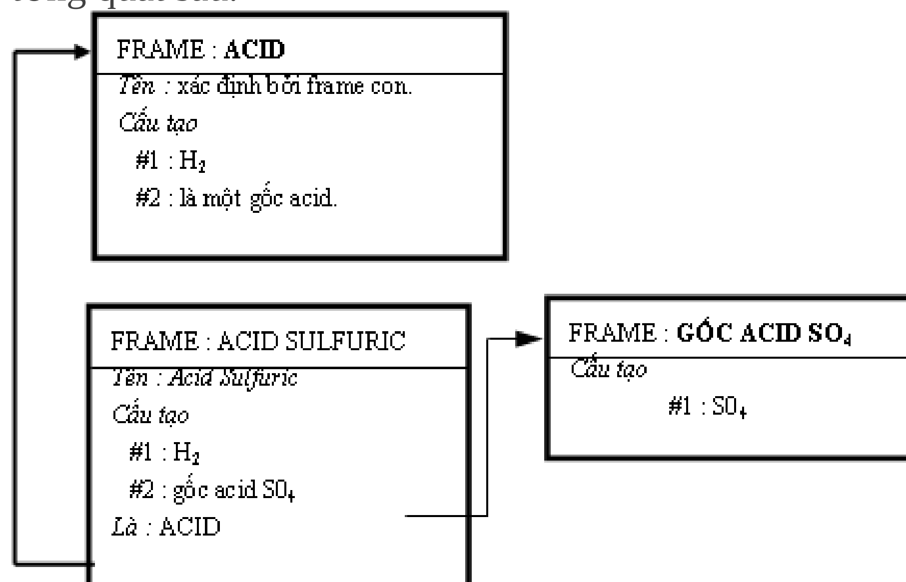
Tuy nhiên, số lượng các phản ứng là rất lớn, nên ta không thể sử dụng các luật dựa trên các phản ứng cụ thể như vậy mà phải sử dụng các phản ứng tổng quát hơn như :

Axit + Bazơ   Muối + Nước

Kiềm + Nước   Xút + H2

(trong hóa học cũng có nhiều phản ứng rất đặc biệt không thể tổng quát được, trong trường hợp này, ta sẽ xem phản ứng đó như là một luật riêng!).

Để mô tả được các phản ứng tổng quát như trên, ta sẽ sử dụng các frame. Chẳng hạn để đặc tả Acid Sulfuric H2SO4 ta sử dụng các frame tổng quát sau.



Dĩ nhiên là trong các frame ở trên còn rất nhiều thuộc tính hóa học khác. Ở đây chúng tôi chỉ trình bày sơ lược về mặt ý tưởng để bạn đọc có cơ sở bắt đầu. Ý tưởng này đã được một số sinh viên năm 4 của khoa Công Nghệ Thông Tin Đại Học Khoa Học Tự Nhiên TP. Hồ Chí Minh cài đặt thành công. Chương trình chạy tốt trong phạm vi các phản ứng trong sách giáo khoa lớp 10, 11 và 12.

Chương 3 MỞ ĐẦU VỀ QUAN MÁY HỌC

I. THẾ NÀO LÀ MÁY HỌC ?

Thuật ngữ "học" theo nghĩa thông thường là tiếp thu tri thức để biết cách vận dụng. Ở ngoài đời, quá trì học diễn ra dưới nhiều hình thức khác nhau như học thuộc lòng (học vẹt), học theo kinh nghiệm (học dựa theo trường hợp), học theo kiểu nghe nhìn,... Trên máy tính cũng có nhiều thuật toán học khác nhau. Tuy nhiên, trong phạm vi của giáo trình này, chúng ta chỉ khảo sát phương pháp học dựa theo trường hợp. Theo phương pháp này, hệ thống sẽ được cung cấp một số các trường hợp "mẫu", dựa trên tập mẫu này, hệ thống sẽ tiến hành phân tích và rút ra các quy luật (biểu diễn bằng luật sinh). Sau đó, hệ thống sẽ dựa trên các luật này để "đánh giá" các trường hợp khác (thường không giống như các trường hợp "mẫu"). Ngay cả chỉ với kiểu học này, chúng ta cũng đã có

nhiều thuật toán học khác nhau. Một lần nữa, với mục đích giới thiệu, chúng ta chỉ khảo sát một trường hợp đơn giản.

Có thể khái quát quá trình học theo trường hợp dưới dạng hình thức như sau :

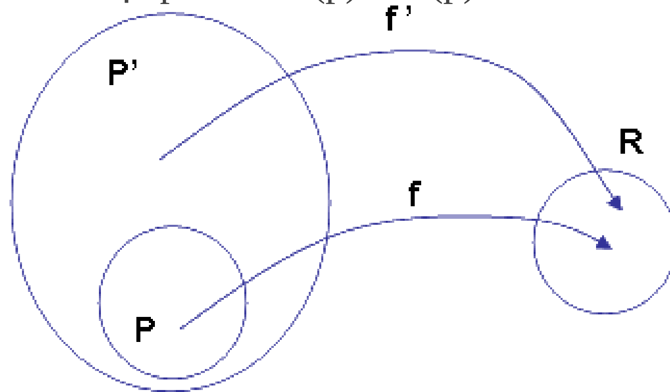
Dữ liệu cung cấp cho hệ thống là một ánh xạ f trong đó ứng một trường hợp p trong tập hợp P với một "lớp" r trong tập R.

f : P | R

p r

Tuy nhiên, tập P thường nhỏ (và hữu hạn) so với tập tất cả các trường hợp cần quan tâm P' (P P'). Mục tiêu của chúng ta là xây dựng ánh xạ f ' sao cho có thể ứng mọi trường hợp p' trong tập P' với một "lớp" r trong tập R. Hơn nữa, f ' phải bảo toàn f, nghĩa là :

Với mọi p P thì f(p) f '(p)



Hình 3.1 : Học theo trường hợp là tìm cách xây dựng ánh xạ f' dựa theo ánh xạ f. f được gọi là tập mẫu.

Phương pháp học theo trường hợp là một phương pháp phổ biến trong cả nghiên cứu khoa học và mê tín dị đoan. Cả hai đều dựa trên các dữ liệu quan sát, thống kê để từ đó rút ra các quy luật. Tuy nhiên, khác với khoa

học, mê tín dị đoan thường dựa trên tập mẫu không đặc trưng, cục bộ, thiếu cơ sở khoa học.

II. HỌC BẰNG CÁCH XÂY DỰNG CÂY ĐỊNH DANH

Phát biểu hình thức có thể khó hình dung. Để cụ thể hơn, ta hãy cùng nhau quan sát một ví dụ cụ. Nhiệm vụ của chúng ta trong ví dụ này là xây dựng các quy luật để có thể kết luận một người như thế nào khi đi tắm biển thì bị cháy nắng. Ta gọi tính chất cháy nắng hay không cháy nắng là thuộc tính quan tâm (thuộc tính mục tiêu). Như vậy, trong trường hợp này, tập R của chúng ta chỉ gồm có hai phần tử {"cháy nắng", "bình thường"}. Còn tập P là tất cả những người được liệt kê trong bảng dưới (8 người) Chúng ta quan sát hiện tượng cháy nắng dựa trên 4 thuộc tính sau : chiều cao (cao, trung bình, thấp), màu tóc (vàng, nâu, đỏ) cân nặng (nhẹ, TB, nặng), dùng kem (có, không),. Ta gọi các thuộc tính này gọi là thuộc tính dẫn xuất.

Dĩ nhiên là trong thực tế để có thể đưa ra được một kết luận như vậy, chúng ta cần nhiều dữ liệu hơn và đồng thời cũng cần nhiều thuộc tính dẫn xuất trên. Ví dụ đơn giản này chỉ nhằm để minh họa ý tưởng của thuật toán máy học mà chúng ta sắp trình bày.

Tên	Tóc	Ch.Cao	Cân Nặng	Dùng kem?	Kết quả
Sarah	Vàng	T.Bình	Nhẹ	Không	Cháy
Dana	Vàng	Cao	T.Bình	Có	Không
Alex	Nâu	Thấp	T.Bình	Có	Không
Annie	Vàng	Thấp	T.Bình	Không	Cháy

Emilie	Đỏ	T.Bình	Nặng	Không	Cháy
Peter	Nâu	Cao	Nặng	Không	Không
John	Nâu	T.Bình	Nặng	Không	Không
Kartie	Vàng	Thấp	Nhẹ	Có	Không

Ý tưởng đầu tiên của phương pháp này là tìm cách phân hoạch tập P ban đầu thành các tập Pi sao cho tất cả các phần tử trong tất cả các tập Pi đều có chung thuộc tính mục tiêu.

P = P1   P2   ...   Pn và   (i,j) i   j : thì (Pi   Pj =   ) và

  i,   k,l : pk   Pi và pl   Pj thì f(pk) = f(pl)

Sau khi đã phân hoạch xong tập P thành tập các phân hoạch Pi được đặc trưng bởi thuộc tính đích ri (ri   R), bước tiếp theo là ứng với mỗi phân hoạch Pi ta xây dựng luật Li : GTi   ri trong đó các GTi là mệnh đề được hình thành bằng cách kết hợp các thuộc tính dẫn xuất.

Một lần nữa, vấn đề hình thức có thể làm bạn cảm thấy khó khăn. Chúng ta hãy thử ý tưởng trên với bảng số liệu mà ta đã có.

Có hai cách phân hoạch hiển nhiên nhất mà ai cũng có thể nghĩ ra. Cách đầu tiên là cho mỗi người vào một phân hoạch riêng (P1 = {Sarah}, P2 = {Dana}, … tổng cộng sẽ có 8 phân hoạch cho 8 người). Cách thứ hai là phân hoạch thành hai tập, một tập gồm tất cả những người cháy nắng và tập còn lại bao gồm tất cả những người không cháy nắng. Tuy đơn giản nhưng phân hoạch theo kiểu này thì chúng ta chẳng giải quyết được gì !!

II.1. Đâm chồi

Chúng ta hãy thử một phương pháp khác. Bây giờ bạn hãy quan sát thuộc tính đầu tiên – màu tóc. Nếu dựa theo màu tóc để phân chia ta sẽ có được 3 phân hoạch khác nhau ứng với mỗi giá trị của thuộc tính màu tóc. Cụ thể là :

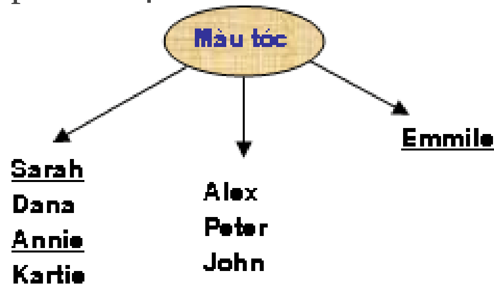
Pvàng = { Sarah, Dana, Annie, Kartie }

Pnâu = { Alex, Peter, John }

Pđỏ = { Emmile }

* Các người bị cháy nắng được gạch dưới và in đậm.

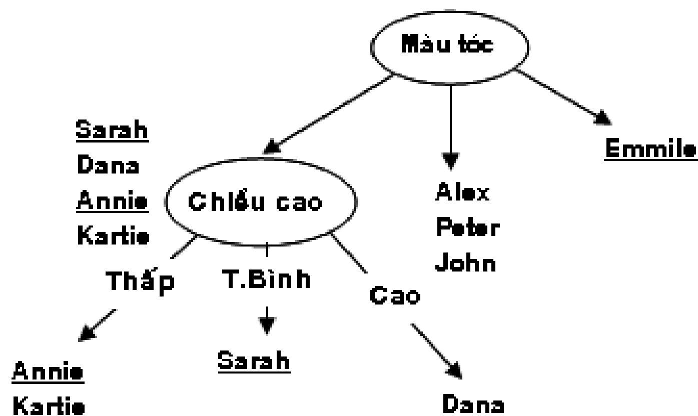
Thay vì liệt kê ra như trên, ta dùng sơ đồ cây để tiện mô tả cho các bước phân hoạch sau :



Quan sát hình trên ta thấy rằng phân hoạch Pnâu và Pđỏ thỏa mãn được điều kiện "có chung thuộc tính mục tiêu" (Pnâu chứa toàn người không cháy nắng, Pđỏ chứa toàn người cháy nắng).

Còn lại tập Pvàng là còn lẫn lộn người cháy năng và không cháy nắng. Ta sẽ tiếp tục phân hoạch tập này thành các tập con. Bây giờ ta hãy quan sát thuộc tính chiều cao. Thuộc tính này giúp phân hoạch tập Pvàng thành 3 tập con : PVàng, Thấp = {Annie, Kartie}, PVàng, T.Bình= {Sarah} và PVàng,Cao= { Dana }

Nếu nối tiếp vào cây ở hình trước ta sẽ có hình ảnh cây phân hoạch như sau :



Quá trình này cứ thế tiếp tục cho đến khi tất cả các nút lá của cây không còn lẫn lộn giữa cháy nắng và không cháy nắng nữa. Bạn cũng thấy rằng, qua mỗi bước phân hoạch cây phân hoạch ngày càng "phình" ra. Chính vì vậy mà quá trình này được gọi là quá trình "đâm chồi". Cây mà chúng ta đang xây dựng được gọi là cây định danh.

Đến đây, chúng ta lại gặp một vấn đề mới. Nếu như ban đầu ta không chọn thuộc tính màu tóc để phân hoạch mà chọn thuộc tính khác như chiều cao chẳng hạn để phân hoạch thì sao? Cuối cùng thì cách phân hoạch nào sẽ tốt hơn?

II.2. Phương án chọn thuộc tính phân hoạch

Vấn đề mà chúng ta gặp phải cũng tương tự như bài toán tìm kiếm : "Đứng trước một ngã rẽ, ta cần phải đi vào hướng nào?". Hai phương pháp đánh giá dưới đây sẽ giúp ta chọn được thuộc tính phân hoạch tại mỗi bước xây dựng cây định danh.

II.2.1. Quinlan

Quinlan quyết định thuộc tính phân hoạch bằng cách xây dựng các vector đặc trưng cho mỗi giá trị của từng thuộc tính dẫn xuất và thuộc tính mục tiêu. Cách tính cụ thể như sau :

Với mỗi thuộc tính dẫn xuất A còn có thể sử dụng để phân hoạch, tính :

VA(j) = (T(j , r1), T(j , r2) , …, T(j , rn))

T(j, ri) = (tổng số phần tử trong phân hoạch có giá trị thuộc tính dẫn xuất A là j và có giá trị thuộc tính mục tiêu là ri) / (tổng số phần tử trong phân hoạch có giá trị thuộc tính dẫn xuất A là j)

* trong đó r1, r2, … , rn là các giá trị của thuộc tính mục tiêu

*

$$\sum_i T(j, r_i) = 1$$

Như vậy nếu một thuộc tính A có thể nhận một trong 5 giá trị khác nhau thì nó sẽ có 5 vector đặc trưng.

Một vector V(Aj) được gọi là vector đơn vị nếu nó chỉ có duy nhất một thành phần có giá trị 1 và những thành phần khác có giá trị 0.

Thuộc tính được chọn để phân hoạch là thuộc tính có nhiều vector đơn vị nhất.

Trở lại ví dụ của chúng ta, ở trạng thái ban đầu (chưa phân hoạch) chúng ta sẽ tính vector đặc trưng cho từng thuộc tính dẫn xuất để tìm ra thuộc tính dùng để phân hoạch. Đầu tiên là thuộc tính màu tóc. Thuộc tính màu tóc có 3 giá trị khác nhau (vàng, đỏ, nâu) nên sẽ có 3 vector đặc trưng tương ứng là :

VTóc (vàng) = (T(vàng, cháy nắng), T(vàng, không cháy nắng))

Số người tóc vàng là : 4

Số người tóc vàng và cháy nắng là : 2

Số người tóc vàng và không cháy nắng là : 2

Do đó

VTóc(vàng) = (2/4 , 2/4) = (0.5, 0.5)

Tương tự

VTóc(nâu) = (0/3, 3/3) = (0,1) (vector đơn vị)

Số người tóc nâu là : 3

Số người tóc nâu và cháy nắng là : 0

Số người tóc nâu và không cháy nắng là : 3

VTóc(đỏ) = (1/1, 0/1) = (1,0) (vector đơn vị)

Tổng số vector đơn vị của thuộc tính tóc vàng là 2

Các thuộc tính khác được tính tương tự, kết quả như sau :

VC.Cao(Cao) = (0/2,2/2) = (0,1)

VC.Cao(T.B) = (2/3,1/3)

VC.Cao(Thấp) = (1/3,2/3)

VC.Nặng (Nhẹ) = (1/2,1/2)

VC.Nặng (T.B) = (1/3,2/3)

VC.Nặng (Nặng) = (1/3,2/3)

VKem (Có) = (3/3,0/3) = (1,0)

VKem (Không) = (3/5,2/5)

Như vậy thuộc tính màu tóc có số vector đơn vị nhiều nhất nên sẽ được chọn để phân hoạch.

Sau khi phân hoạch theo màu tóc xong, chỉ có phân hoạch theo tóc vàng (Pvàng) là còn chứa những người cháy nắng và không cháy nắng nên ta sẽ tiếp tục phân hoạch tập này. Ta sẽ thực hiện thao tác tính vector đặc trưng tương tự đối với các thuộc tính còn lại (chiều cao, cân nặng, dùng kem). Trong phân hoạch Pvàng, tập dữ liệu của chúng ta còn lại là :

Tên	Ch.Cao	Cân Nặng	Dùng kem?	Kết quả
Sarah	T.Bình	Nhẹ	Không	Cháy
Dana	Cao	T.Bình	Có	Không
Annie	Thấp	T.Bình	Không	Cháy
Kartie	Thấp	Nhẹ	Có	Không

VC.Cao(Cao) = (0/1,1/1) = (0,1)

VC.Cao(T.B) = (1/1,0/1) = (1,0)

VC.Cao(Thấp) = (1/2,1/2)

VC.Nặng (Nhẹ) = (1/2,1/2)

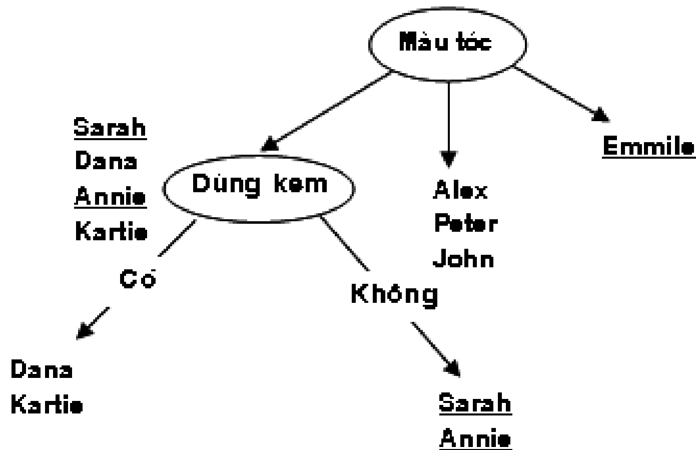
VC.Nặng (T.B) = (1/2,1/2)

VC.Nặng (Nặng) = (0,0)

VKem (Có) = (0/2,2/2) = (0,1)

VKem (Không) = (2/2,0/2) = (1,0)

2 thuộc tính dùmg kem và chiều cao đều có 2 vector đơn vị. Tuy nhiên, số phân hoạch của thuộc tính dùng kem là ít hơn nên ta chọn phân hoạch theo thuộc tính dùng kem. Cây định danh cuối cùng của chúng ta sẽ như sau :



II.2.2. Độ đo hỗn loạn

Thay vì phải xây dựng các vector đặc trưng như phương pháp của Quinlan, ứng với mỗi thuộc tính dẫn xuất ta chỉ cần tính ra độ đo hỗn loạn và lựa chọn thuộc tính nào có độ đo hỗn loại là thấp nhất. Công thức tính như sau :

TA =

$$\sum_j \left(\frac{b_j}{b_t} \times \sum_i \left(-\frac{b_{ri}}{b_j} \times \log_2 \left(-\frac{b_{ri}}{b_j} \right) \right) \right)$$

trong đó :

bt là tổng số phần tử có trong phân hoạch

bj là tổng số phần tử có thuộc tính dẫn xuất A có giá trị j.

bri : tổng số phần tử có thuộc tính dẫn xuất A có giá trị j và thuộc tính mục tiêu có giá trị i.

II.3. Phát sinh tập luật

Nguyên tắc phát sinh tập luật từ cây định danh khá đơn giản. Ứng với mỗi nút lá, ta chỉ việc đi từ đỉnh cho đến nút lá đó và phát sinh ra luật tương ứng. Cụ thể là từ cây định danh kết quả ở cuối phần II.2 ta có các luật sau (xét các nút lá từ trái sang phải)

(Màu tóc vàng) và (có dùng kem)   không cháy nắng

(Màu tóc vàng) và (không dùng kem)   cháy nắng

(Màu tóc nâu)   không cháy nắng

(Màu tóc đỏ)   cháy nắng

Khá đơn giản phải không? Có lẽ không có gì phải nói gì thêm. Chúng ta hãy thực hiện bước cuối cùng là tối ưu tập luật.

II.4. Tối ưu tập luật

II.4.1. Loại bỏ mệnh đề thừa

Khác so với các phương pháp loại bỏ mệnh đề thừa đã được trình bày trong phần biểu diễn tri thức (chỉ quan tâm đến logic hình thức), phương pháp loại bỏ mệnh đề thừa ở đây dựa vào dữ liệu. Với ví dụ và tập luật đã có ở phần trước, bạn hãy quan sát luật sau :

(Màu tóc vàng) và (có dùng kem)   không cháy nắng

Bây giờ ta hãy lập một bảng (gọi là bảng Contigency), bảng thống kê những người có dùng kem tương ứng với tóc màu vàng và bị cháy nắng

hay không. Trong dữ liệu đã cho, có 3 người không dùng kem.

	Không cháy nắng	Cháy nắng
Màu vàng	2	0
Màu khác	1	0

Theo bảng thống kê này thì rõ ràng là thuộc tính tóc vàng (trong luật trên) không đóng góp gì trong việc đưa ra kết luận cháy nắng hay không (cả 3 người dùng kem đều không cháy nắng) nên ta có thể loại bỏ thuộc tính tóc vàng ra khỏi tập luật.

Sau khi loại bỏ mệnh đề thừa, tập mệnh đề của chúng ta trong ví dụ trên sẽ còn :

(có dùng kem)   không cháy nắng

(Màu tóc vàng) và (không dùng kem)   cháy nắng

(Màu tóc nâu)   không cháy nắng

(Màu tóc đỏ)   cháy nắng

Như vậy quy tắc chung để có thể loại bỏ một mệnh đề là như thế nào? Rất đơn giản, giả sử luật của chúng ta có n mệnh đề :

A1 và A2 và … và An   R

Để kiểm tra xem có thể loại bỏ mệnh đề Ai hay không, bạn hãy lập ra một tập hợp P bao gồm các phần tử thỏa tất cả mệnh đề A1 , A2 , … Ai-, Ai+1, …, An (lưu ý : không cần xét là có thỏa Ai hay không, chỉ cần thỏa các mệnh đề còn lại là được)

Sau đó, bạn hãy lập bảng Contigency như sau :

	R	R
Ai	E	F
Ai	G	H

Trong đó

E là số phần tử trong P thỏa cả Ai và R.

F là số phần tử trong P thỏa Ai và không thỏa R

G là số phần tử trong P không thỏa Ai và thỏa R

H là số phần tử trong P không thỏa Ai và không thỏa R

Nếu tổng F+H = 0 thì có thể loại bỏ mệnh đề Ai ra khỏi luật.

II.4.2. Xây dựng mệnh đề mặc định

Có một vấn đề đặt ra là khi gặp phải một trường hợp mà tất cả các luật đều không thỏa thì phải làm như thế nào? Một cách hành động là đặt ra một luật mặc định đại loại như :

Nếu không có luật nào thỏa   cháy nắng (1)

Hoặc

Nếu không có luật nào thỏa   không cháy nắng. (2)

(chỉ có hai luật vì thuộc tính mục tiêu chỉ có thể nhận một trong hai giá trị là cháy nắng hay không cháy nắng)

Giả sử ta đã chọn luật mặc định là (2) thì tập luật của chúng ta sẽ trở thành :

(Màu tóc vàng) và (không dùng kem)   cháy nắng

(Màu tóc đỏ)   cháy nắng

Nếu không có luật nào thỏa   không cháy nắng. (2)

Lưu ý rằng là chúng ta đã loại bỏ đi tất cả các luật dẫn đến kết luận không cháy nắng và thay nó bằng luật mặc định. Tại sao vậy? Bởi vì các luật này có cùng kết luận với luật mặc định. Rõ ràng là chỉ có thể có một trong hai khả năng là cháy nắng hay không.

Vấn đề là chọn luật nào? Sau đây là một số quy tắc.

1) Chọn luật mặc định sao cho nó có thể thay thế cho nhiều luật nhất. (trong ví dụ của ta thì nguyên tắc này không áp dụng được vì có 2 luật dẫn đến cháy nắng và 2 luật dẫn đến không cháy nắng)

2) Chọn luật mặc định có kết luận phổ biến nhất. Trong ví dụ của chúng ta thì nên chọn luật (2) vì số trường hợp không cháy nắng là 5 còn không cháy nắng là 3.

3) Chọn luật mặc định sao cho tổng số mệnh đề của các luật mà nó thay thế là nhiều nhất. Trong ví dụ của chúng ta thì luật được chọn sẽ là luật (1) vì tổng số mệnh đề của luật dẫn đến cháy nắng là 3 trong khi tổng số mệnh đề của luật dẫn đến không cháy nắng chỉ là 2.

BÀI TẬP

CHƯƠNG 1

1) Viết chương trình giải bài toán hành trình người bán hàng rong bằng hai thuật giải GTS1 và GTS2 trong trường hợp có n địa điểm khác nhau.

2) Viết chương trình giải bài toán phân công công việc bằng cách ứng dụng nguyên lý thứ tự.

3) Ứng dụng nguyên lý thứ tự, hãy giải bài toán chia đồ vật sau. Có n vật với khối lượng lần lượt là M1, M2, … Mn. Hãy tìm cách chia n vật này thành hai nhóm sao cho chênh lệch khối lượng giữa hai nhóm này là nhỏ nhất.

4) Viết chương trình giải bài toán mã đi tuần.

5) Viết chương trình giải bài toán 8 hậu.

6) Viết chương trình giải bài toán Ta-canh bằng thuật giải A*.

7) Viết chương trình giải bài toán tháp Hà Nội bằng thuật giải A*.

8)* Viết chương trình tìm kiếm đường đi ngắn nhất trong một bản đồ tổng quát. Bản đồ được biểu diễn bằng một mảng hai chiều A, trong đó A[x,y]=0 là có thể đi được và A[x,y]= 1 là vật cản. Cho phép người dùng click chuột trên màn hình để tạo bản đồ và xác định điểm xuất phát và kết thúc. Chi phí để đi từ một ô bất kỳ sang ô kế cận nó là 1.

Mở rộng bài toán trong trường hợp chi phí để di chuyển từ ô (x,y) sang một bất kỳ kế (x,y) là A[x,y].

CHƯƠNG 2

1. Viết chương trình minh họa các bước giải bài toán đong nước (sử dụng đồ họa càng tốt).
2. Viết chương trình cài đặt hai thuật toán Vương Hạo và Robinson trong đó liệt kê các bước chứng minh một biểu thức logic.

3. Viết chương trình giải bài toán tam giác tổng quát bằng mạng ngữ nghĩa (lưu ý sử dụng thuật toán ký pháp nghịch đảo Ba Lan)
4. Hãy thử xây dựng một bộ luật phức tạp hơn trong ví dụ đã được trình bày dùng để chuẩn đoán hỏng hóc của máy tính. Viết chương trình ứng dụng bộ luật này trong việc chuẩn đoán hỏng hóc của máy tính (sử dùng thuật toán suy diễn lùi).
5. Hãy cài đặt các frame đặc tả các đối tượng hình học bằng kỹ thuật hướng đối tượng trong ngôn ngữ lập trình mà bạn quen dùng. Hãy xây dựng một ngôn ngữ script đơn giản cho phép người dùng có thể sử dụng các frame này trong việc giải một số bài toán hình học đơn giản.

CHƯƠNG 3

1) Cho bảng số liệu sau

Hãy xây dựng cây định danh và tìm luật để xác định một người là Châu Âu hay Châu Á bằng hai phương pháp vector đặc trưng của Quinlan và độ đo hỗn loạn.

STT	Dáng	Cao	Giới	Châu
1	To	TB	Nam	Á
2	Nhỏ	Cao	Nam	Á
3	Nhỏ	TB	Nam	Âu
4	To	Cao	Nam	Âu
5	Nhỏ	TB	Nữ	Âu

6	Nhỏ	Cao	Nam	Âu
7	Nhỏ	Cao	Nữ	Âu
8	To	TB	Nữ	Âu

2)* Viết chương trình cài đặt tổng quát thuật toán học dựa trên việc xây dựng cây định danh. Chương trình yêu cầu người dùng đưa vào danh sách các thuộc tính dẫn xuất, thuộc tính mục tiêu cùng với tất cả các giá trị của mỗi thuộc tính; yêu cầu người dùng cung cấp bảng số liệu quan sát. Chương trình sẽ liệt kê lên màn hình các luật mà nó tìm được từ bảng số liệu. Sau đó, yêu cầu người dùng nhập vào các trường hợp cần xác định, hệ thống sẽ đưa ra kết luận của trường hợp này.

Lưu ý : Nên sử dụng một hệ quản trị CSDL để cài đặt chương trình này.

GS.TSKH. Hoàng KiếmThs. Đinh Nguyễn Anh Dũng

Tổng quan về Ngôn ngữ lập trình
Môn Lập Trình Căn Bản A cung cấp cho sinh viên những kiến thức cơ bản về lập trình thông qua ngôn ngữ lập trình C. Môn học này là nền tảng để tiếp thu hầu hết các môn học khác trong chương trình đào tạo. Mặt khác, nắm vững ngôn ngữ C là cơ sở để phát triển các ứng dụng. Học xong môn này, sinh viên phải nắm được các vấn đề sau: - Khái niệm về ngôn ngữ lập trình. - Khái niệm về kiểu dữ liệu - Kiểu dữ liệu có cấu trúc (cấu trúc dữ liệu). - Khái niệm về giải thuật - Ngôn ngữ biểu diễn giải thuật. - Ngôn ngữ sơ đồ (lưu đồ), sử dụng lưu đồ để biểu diễn các giải thuật. - Tổng quan về Ngôn ngữ lập trình C. - Các kiểu dữ liệu trong C. - Các lệnh có cấu trúc. - Cách thiết kế và sử dụng các hàm trong C. - Một số cấu trúc dữ liệu trong C.

ĐỐI TƯỢNG MÔN HỌC

Môn học lập trình căn bản được dùng để giảng dạy cho các sinh viên sau:

- Sinh viên năm thứ 2 chuyên ngành Tin học, Toán Tin, Lý Tin.

- Sinh viên năm thứ 2 chuyên ngành Điện tử (Viễn thông, Tự động hóa…)

NỘI DUNG CỐT LÕI

Trong khuôn khổ 45 tiết, giáo trình được cấu trúc thành 2 phần: Phần 1 giới thiệu về lập trình cấu trúc, các khái niệm về lập trình, giải thuật… Phần 2 trình bày có hệ thống về ngôn ngữ lập trình C, các câu lệnh, các kiểu dữ liệu…

PHẦN 1: Giới thiệu cấu trúc dữ liệu và giải thuật

PHẦN 2: Giới thiệu về một ngôn ngữ lập trình - Ngôn ngữ lập trình C

Chương 1: Giới thiệu về ngôn ngữ C & môi trường lập trình Turbo C

Chương 2: Các thành phần của ngôn ngữ C

Chương 3: Các kiểu dữ liệu sơ cấp chuẩn và các lệnh đơn

Chương 4: Các lệnh có cấu trúc

Chương 5: Chương trình con

Chương 6: Kiểu mảng

Chương 7: Kiểu con trỏ

Chương 8: Kiểu chuỗi ký tự

Chương 9: Kiểu cấu trúc

Chương 10: Kiểu tập tin

KIẾN THỨC LIÊN QUAN

Để học tốt môn Lập Trình Căn Bản A, sinh viên cần phải có các kiến thức nền tảng sau:

- Kiến thức toán học.

- Kiến thức và kỹ năng thao tác trên máy tính.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] Nguyễn Văn Linh, Giáo trình Tin Học Đại Cương A, Khoa Công Nghệ Thông Tin, Đại học Cần Thơ, 1991.

[2] Nguyễn Đình Tê, Hoàng Đức Hải , Giáo trình lý thuyết và bài tập ngôn ngữ C; Nhà xuất bản Giáo dục, 1999.

[3] Nguyễn Cẩn, C – Tham khảo toàn diện, Nhà xuất bản Đồng Nai, 1996.

[4] Võ Văn Viện, Giúp tự học Lập Trình với ngôn ngữ C, Nhà xuất bản Đồng Nai, 2002.

[5] Brain W. Kernighan & Dennis Ritchie, The C Programming Language, Prentice Hall Publisher, 1988.

TỪ KHÓA

Bài toán, chương trình, giải thuật, ngôn ngữ giả, lưu đồ, biểu thức, gán, rẽ nhánh, lặp, hàm, mảng, con trỏ, cấu trúc, tập tin.

Các thành phần cơ bản của ngôn ngữ C
Học xong chương này, sinh viên sẽ nắm được các vấn đề sau: - Bộ chữ viết trong C. - Các từ khóa. - Danh biểu. - Các kiểu dữ liệu - Biến và các biểu thức trong C. - Cấu trúc của một chương trình viết bằng ngôn ngữ lập trình C

Bộ chữ viết trong C

Bộ chữ viết trong ngôn ngữ C bao gồm những ký tự, ký hiệu sau: (phân biệt chữ in hoa và in thường):

- 26 chữ cái latinh lớn A,B,C...Z
- 26 chữ cái latinh nhỏ a,b,c ...z.
- 10 chữ số thập phân 0,1,2...9.
- Các ký hiệu toán học: +, -, *, /, =, <, >, (,)
- Các ký hiệu đặc biệt: :. , ; " ' _ @ # $! ^ [ ] { } ...
- Dấu cách hay khoảng trống.

Các từ khoá trong C

Từ khóa là các từ dành riêng (reserved words) của C mà người lập trình có thể sử dụng nó trong chương trình tùy theo ý nghĩa của từng từ. Ta không được dùng từ khóa để đặt cho các tên của riêng mình. Các từ khóa của Turbo C 3.0 bao gồm:

asm auto break case cdecl char

class const continue _cs default delete

do double _ds else enum _es

extern _export far _fastcall float for

friend goto huge if inline int

interrupt _loadds long near new operator

pascal private protected public register return

_saveregs _seg short signed sizeof _ss

static struct switch template this typedef

union unsigned virtual void volatile while

Cặp dấu ghi chú thích

Khi viết chương trình đôi lúc ta cần phải có vài lời ghi chú về 1 đoạn chương trình nào đó để dễ nhớ và dễ điều chỉnh sau này; nhất là phần nội dung ghi chú phải không thuộc về chương trình (khi biên dịch phần này bị bỏ qua). Trong ngôn ngữ lập trình C, nội dung chú thích phải được viết trong cặp dấu /* và */.

Ví dụ :

```
#include <stdio.h>

#include<conio.h>

int main ()

{

char ten[50]; /* khai bao bien ten kieu char 50 ky tu */

/*Xuat chuoi ra man hinh*/

printf(“Xin cho biet ten cua ban !”);

scanf(“%s”,ten); /*Doc vao 1 chuoi la ten cua ban*/

printf(“Xin chao ban %s\n ”,ten);

printf(“Chao mung ban den voi Ngon ngu lap trinh C”);
```

```
/*Dung chuong trinh, cho go phim*/
getch();
return 0;
}
```

CÁC KIỂU DỮ LIỆU SƠ CẤP CHUẨN TRONG C

Các kiểu dữ liệu sơ cấp chuẩn trong C có thể được chia làm 2 dạng : kiểu số nguyên, kiểu số thực.

Kiểu số nguyên

Kiểu số nguyên là kiểu dữ liệu dùng để lưu các giá trị nguyên hay còn gọi là kiểu đếm được. Kiểu số nguyên trong C được chia thành các kiểu dữ liệu con, mỗi kiểu có một miền giá trị khác nhau

Kiểu số nguyên 1 byte (8 bits)

Kiểu số nguyên một byte gồm có 2 kiểu sau:

STT	Kiểu dữ liệu	Miền giá trị (Domain)
1	unsigned char	Từ 0 đến 255 (tương đương 256 ký tự trong bảng mã ASCII)

2	char	Từ -128 đến 127

Kiểu unsigned char: lưu các số nguyên dương từ 0 đến 255.

=> Để khai báo một biến là kiểu ký tự thì ta khai báo biến kiểu unsigned char. Mỗi số trong miền giá trị của kiểu unsigned char tương ứng với một ký tự trong bảng mã ASCII .

Kiểu char: lưu các số nguyên từ -128 đến 127. Kiểu char sử dụng bit trái nhất để làm bit dấu.

=> Nếu gán giá trị > 127 cho biến kiểu char thì giá trị của biến này có thể là số âm (?).

Kiểu số nguyên 2 bytes (16 bits)

Kiểu số nguyên 2 bytes gồm có 4 kiểu sau:

STT	Kiểu dữ liệu	Miền giá trị (Domain)
1	enum	Từ -32,768 đến 32,767
2	unsigned int	Từ 0 đến 65,535
3	short int	Từ -32,768 đến 32,767
4	int	Từ -32,768 đến 32,767

Kiểu enum, short int, int : Lưu các số nguyên từ -32768 đến 32767. Sử dụng bit bên trái nhất để làm bit dấu.

=> Nếu gán giá trị >32767 cho biến có 1 trong 3 kiểu trên thì giá trị của biến này có thể là số âm.

Kiểu unsigned int: Kiểu unsigned int lưu các số nguyên dương từ 0 đến 65535.

Kiểu số nguyên 4 byte (32 bits)

Kiểu số nguyên 4 bytes hay còn gọi là số nguyên dài (long) gồm có 2 kiểu sau:

STT	Kiểu dữ liệu	Miền giá trị (Domain)
1	unsigned long	Từ 0 đến 4,294,967,295
2	long	Từ -2,147,483,648 đến 2,147,483,647

Kiểu long : Lưu các số nguyên từ -2147483658 đến 2147483647. Sử dụng bit bên trái nhất để làm bit dấu.

=> Nếu gán giá trị >2147483647 cho biến có kiểu long thì giá trị của biến này có thể là số âm.

Kiểu unsigned long: Kiểu unsigned long lưu các số nguyên dương từ 0 đến 4294967295

Kiểu số thực

Kiểu số thực dùng để lưu các số thực hay các số có dấu chấm thập phân gồm có 3 kiểu sau:

STT	Kiểu dữ liệu	Kích thước (Size)	Miền giá trị (Domain)
1	float	4 bytes	Từ 3.4 * 10-38 đến 3.4 * 1038
2	double	8 bytes	Từ 1.7 * 10-308 đến 1.7 * 10308
3	long double	10 bytes	Từ 3.4 *10-4932 đến 1.1 *104932

Mỗi kiểu số thực ở trên đều có miền giá trị và độ chính xác (số số lẻ) khác nhau. Tùy vào nhu cầu sử dụng mà ta có thể khai báo biến thuộc 1 trong 3 kiểu trên.

Ngoài ra ta còn có kiểu dữ liệu void, kiểu này mang ý nghĩa là kiểu rỗng không chứa giá trị gì cả.

Tên và hằng trong C

Tên (danh biểu)

Tên hay còn gọi là danh biểu (identifier) được dùng để đặt cho chương trình, hằng, kiểu, biến, chương trình con... Tên có hai loại là tên chuẩn và tên do người lập trình đặt.

Tên chuẩn là tên do C đặt sẵn như tên kiểu: int, char, float,…; tên hàm: sin, cos...

Tên do người lập trình tự đặt để dùng trong chương trình của mình. Sử dụng bộ chữ cái, chữ số và dấu gạch dưới (_) để đặt tên, nhưng phải tuân thủ quy tắc:

- Bắt đầu bằng một chữ cái hoặc dấu gạch dưới.
- Không có khoảng trống ở giữa tên.
- Không được trùng với từ khóa.
- Độ dài tối đa của tên là không giới hạn, tuy nhiên chỉ có 31 ký tự đầu tiên là có ý nghĩa.
- Không cấm việc đặt tên trùng với tên chuẩn nhưng khi đó ý nghĩa của tên chuẩn không còn giá trị nữa.

Ví dụ: tên do người lập trình đặt: Chieu_dai, Chieu_Rong, Chu_Vi, Dien_Tich

Tên không hợp lệ: Do Dai, 12A2,…

Hằng (Constant)

Là đại lượng không đổi trong suốt quá trình thực thi của chương trình.

Hằng có thể là một chuỗi ký tự, một ký tự, một con số xác định. Chúng có thể được biểu diễn hay định dạng (Format) với nhiều dạng thức khác nhau.

Hằng số thực

Số thực bao gồm các giá trị kiểu float, double, long double được thể hiện theo 2 cách sau:

- Cách 1: Sử dụng cách viết thông thường mà chúng ta đã sử dụng trong các môn Toán, Lý, …Điều cần lưu ý là sử dụng dấu thập phân là dấu chấm (.);

Ví dụ: 123.34-223.3333.00-56.0

- Cách 2: Sử dụng cách viết theo số mũ hay số khoa học. Một số thực được tách làm 2 phần, cách nhau bằng ký tự e hay E

Phần giá trị: là một số nguyên hay số thực được viết theo cách 1.

Phần mũ: là một số nguyên

Giá trị của số thực là: Phần giá trị nhân với 10 mũ phần mũ.

Ví dụ: 1234.56e-3= 1.23456 (là số 1234.56 * 10-3)

-123.45E4 = -1234500 (là -123.45 *104)

Hằng số nguyên

Số nguyên gồm các kiểu int (2 bytes) , long (4 bytes) được thể hiện theo những cách sau.

- Hằng số nguyên 2 bytes (int) hệ thập phân: Là kiểu số mà chúng ta sử dụng thông thường, hệ thập phân sử dụng các ký số từ 0 đến 9 để biểu diễn một giá trị nguyên.

Ví dụ: 123 (một trăm hai mươi ba), -242 (trừ hai trăm bốn mươi hai).

- Hằng số nguyên 2 byte (int) hệ bát phân: Là kiểu số nguyên sử dụng 8 ký số từ 0 đến 7 để biểu diễn một số nguyên.

Cách biểu diễn: 0<các ký số từ 0 đến 7>

Ví dụ : 0345 (số 345 trong hệ bát phân)

-020 (số -20 trong hệ bát phân)

Cách tính giá trị thập phân của số bát phân như sau:

Số bát phân : 0dndn-1dn-2…d1d0 (di có giá trị từ 0 đến 7)

=> Giá trị thập phân= $\sum_{i=0}^{n} d_i * 8^i$

0345=229 , 020=16

- Hằng số nguyên 2 byte (int) hệ thập lục phân: Là kiểu số nguyên sử dụng 10 ký số từ 0 đến 9 và 6 ký tự A, B, C, D, E ,F để biểu diễn một số nguyên.

Ký tự giá trị

A10

B11

C12

D13

E14

F15

Cách biểu diễn: 0x<các ký số từ 0 đến 9 và 6 ký tự từ A đến F>

Ví dụ:

0x345 (số 345 trong hệ 16)

0x20 (số 20 trong hệ 16)

0x2A9 (số 2A9 trong hệ 16)

Cách tính giá trị thập phân của số thập lục phân như sau:

Số thập lục phân : 0xdndn-1dn-2…d1d0 (di từ 0 đến 9 hoặc A đến F)

=> Giá trị thập phân= $\sum_{i=0}^{n} d_i * 16^i$

0x345=827 , 0x20=32 , 0x2A9= 681

- Hằng số nguyên 4 byte (long): Số long (số nguyên dài) được biểu diễn như số int trong hệ thập phân và kèm theo ký tự l hoặc L. Một số nguyên nằm ngoài miền giá trị của số int (2 bytes) là số long (4 bytes).

Ví dụ: 45345L hay 45345l hay 45345

- Các hằng số còn lại: Viết như cách viết thông thường (không có dấu phân cách giữa 3 số)

Ví dụ:

12 (mười hai)

12.45 (mười hai chấm 45)

1345.67 (một ba trăm bốn mươi lăm chấm sáu mươi bảy)

Hằng ký tự

Hằng ký tự là một ký tự riêng biệt được viết trong cặp dấu nháy đơn (‘). Mỗi một ký tự tương ứng với một giá trị trong bảng mã ASCII. Hằng ký tự cũng được xem như trị số nguyên.

Ví dụ: ‘a’, ‘A’, ‘0’, ‘9’

Chúng ta có thể thực hiện các phép toán số học trên 2 ký tự (thực chất là thực hiện phép toán trên giá trị ASCII của chúng)

Hằng chuỗi ký tự

Hằng chuỗi ký tự là một chuỗi hay một xâu ký tự được đặt trong cặp dấu nháy kép (“).

Ví dụ: “Ngon ngu lap trinh C”, “Khoa CNTT-DHCT”, “NVLinh-DVHieu”

Chú ý:

1. Một chuỗi không có nội dung “” được gọi là chuỗi rỗng.

2. Khi lưu trữ trong bộ nhớ, một chuỗi được kết thúc bằng ký tự NULL (‘\0’: mã Ascii là 0).

3. Để biểu diễn ký tự đặc biệt bên trong chuỗi ta phải thêm dấu \ phía trước.

Ví dụ: “I’m a student” phải viết “I\’m a student”

“Day la ky tu “dac biet”” phải viết “Day la ky tu \”dac biet\”“

BIẾN VÀ BIỂU THỨC

Biến

Biến là một đại lượng được người lập trình định nghĩa và được đặt tên thông qua việc khai báo biến. Biến dùng để chứa dữ liệu trong quá trình thực hiện chương trình và giá trị của biến có thể bị thay đổi trong quá trình này. Cách đặt tên biến giống như cách đặt tên đã nói trong phần trên.

Mỗi biến thuộc về một kiểu dữ liệu xác định và có giá trị thuộc kiểu đó.

Cú pháp khai báo biến:

<Kiểu dữ liệu> Danh sách các tên biến cách nhau bởi dấu phẩy;

Ví dụ:

int a, b, c; /*Ba biến a, b,c có kiểu int*/

long int chu_vi; /*Biến chu_vi có kiểu long*/

float nua_chu_vi; /*Biến nua_chu_vi có kiểu float*/

double dien_tich; /*Biến dien_tich có kiểu double*/

Lưu ý: Để kết thúc 1 lệnh phải có dấu chấm phẩy (;) ở cuối lệnh.

Vị trí khai báo biến trong C

Trong ngôn ngữ lập trình C, ta phải khai báo biến đúng vị trí. Nếu khai báo (đặt các biến) không đúng vị trí sẽ dẫn đến những sai sót ngoài ý muốn mà người lập trình không lường trước (hiệu ứng lề). Chúng ta có 2 cách đặt vị trí của biến như sau:

a) Khai báo biến ngoài: Các biến này được đặt bên ngoài tất cả các hàm và nó có tác dụng hay ảnh hưởng đến toàn bộ chương trình (còn gọi là biến toàn cục).

Ví dụ:

int i; /*Bien ben ngoai */

float pi; /*Bien ben ngoai*/

int main()

{ … }

b) Khai báo biến trong: Các biến được đặt ở bên trong hàm, chương trình chính hay một khối lệnh. Các biến này chỉ có tác dụng hay ảnh hưởng đến hàm, chương trình hay khối lệnh chứa nó. Khi khai báo biến, phải đặt các biến này ở đầu của khối lệnh, trước các lệnh gán, …

Ví dụ 1:

```
#include <stdio.h>

#include<conio.h>

int bienngoai;/*khai bao bien ngoai*/

int main ()

{ int j,i;/*khai bao bien ben trong chuong trinh chinh*/

clrscr();

i=1; j=2;

bienngoai=3;

printf("\n Gia7 tri cua i la %d",i);

/*%d là số nguyên, sẽ biết sau */

printf("\n Gia tri cua j la %d",j);

printf("\n Gia tri cua bienngoai la %d",bienngoai);

getch();

return 0;

}
```

Ví dụ 2:

```
#include <stdio.h>

#include<conio.h>

int main ()

{ int i, j;/*Bien ben trong*/
```

```
clrscr();

i=4; j=5;

printf("\n Gia tri cua i la %d",i);

printf("\n Gia tri cua j la %d",j);

if(j>i)

{

int hieu=j-i; /*Bien ben trong */

printf("\n Hieu so cua j tru i la %d",hieu);

}

else

{

int hieu=i-j; /*Bien ben trong*/

printf("\n Gia tri cua i tru j la %d",hieu);

}

getch();

return 0;

}
```

Biểu thức

Biểu thức là một sự kết hợp giữa các toán tử (operator) và các toán hạng (operand) theo đúng một trật tự nhất định.

Mỗi toán hạng có thể là một hằng, một biến hoặc một biểu thức khác.

Trong trường hợp, biểu thức có nhiều toán tử, ta dùng cặp dấu ngoặc đơn () để chỉ định toán tử nào được thực hiện trước.

Ví dụ: Biểu thức nghiệm của phương trình bậc hai:

(-b + sqrt(Delta))/(2*a)

Trong đó 2 là hằng; a, b, Delta là biến.

Các toán tử số học

Trong ngôn ngữ C, các toán tử +, -, *, / làm việc tương tự như khi chúng làm việc trong các ngôn ngữ khác. Ta có thể áp dụng chúng cho đa số kiểu dữ liệu có sẵn được cho phép bởi C. Khi ta áp dụng phép / cho một số nguyên hay một ký tự, bất kỳ phần dư nào cũng bị cắt bỏ. Chẳng hạn, 5/2 bằng 2 trong phép chia nguyên.

Toán tử	Ý nghĩa
+	Cộng
-	Trừ
*	Nhân
/	Chia
%	Chia lấy phần dư

--	Giảm 1 đơn vị
++	Tăng 1 đơn vị

Tăng và giảm (++ & --)

Toán tử ++ thêm 1 vào toán hạng của nó và – trừ bớt 1. Nói cách khác:

x = x + 1 giống như ++x

x = x – 1 giống như x—

Cả 2 toán tử tăng và giảm đều có thể tiền tố (đặt trước) hay hậu tố (đặt sau) toán hạng. Ví dụ:x = x + 1 có thể viết x++ (hay ++x)

Tuy nhiên giữa tiền tố và hậu tố có sự khác biệt khi sử dụng trong 1 biểu thức. Khi 1 toán tử tăng hay giảm đứng trước toán hạng của nó, C thực hiện việc tăng hay giảm trước khi lấy giá trị dùng trong biểu thức. Nếu toán tử đi sau toán hạng, C lấy giá trị toán hạng trước khi tăng hay giảm nó. Tóm lại:

x = 10

y = ++x //y = 11

Tuy nhiên:

x = 10

x = x++ //y = 10

Thứ tự ưu tiên của các toán tử số học:

++ -- sau đó là * / % rồi mới đến + -

Các toán tử quan hệ và các toán tử Logic

Ý tưởng chính của toán tử quan hệ và toán tử Logic là đúng hoặc sai. Trong C mọi giá trị khác 0 được gọi là đúng, còn sai là 0. Các biểu thức sử dụng các toán tử quan hệ và Logic trả về 0 nếu sai và trả về 1 nếu đúng.

Toán tử	Ý nghĩa
Các toán tử quan hệ	
>	Lớn hơn
>=	Lớn hơn hoặc bằng
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
==	Bằng
!=	Khác
Các toán tử Logic	
&&	AND
||	OR
!	NOT

Bảng chân trị cho các toán tử Logic:

P	q	p&&q	p||q	!p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Các toán tử quan hệ và Logic đều có độ ưu tiên thấp hơn các toán tử số học. Do đó một biểu thức như: 10 > 1+ 12 sẽ được xem là 10 > (1 + 12) và kết quả là sai (0).

Ta có thể kết hợp vài toán tử lại với nhau thành biểu thức như sau:

10>5&&!(10<9)||3<=4 Kết quả là đúng

Thứ tự ưu tiên của các toán tử quan hệ là Logic

Cao nhất:!

>>=<<=

==!=

&&

Thấp nhất:||

Các toán tử Bitwise:

Các toán tử Bitwise ý nói đến kiểm tra, gán hay sự thay đổi các Bit thật sự trong 1 Byte của Word, mà trong C chuẩn là các kiểu dữ liệu và biến

char, int. Ta không thể sử dụng các toán tử Bitwise với dữ liệu thuộc các kiểu float, double, long double, void hay các kiểu phức tạp khác.

Toán tử	Ý nghĩa
&	AND
|	OR
^	XOR
~	NOT
>>	Dịch phải
<<	Dịch trái

Bảng chân trị của toán tử ^ (XOR)

p	q	p^q
0	0	0
0	1	1
1	0	1

1	1	0

Toán tử ? cùng với :

C có một toán tử rất mạnh và thích hợp để thay thế cho các câu lệnh của If-Then-Else. Cú pháp của việc sử dụng toán tử ? là:

E1?E2:E3

Trong đó E1, E2, E3 là các biểu thức.

Ý nghĩa: Trước tiên E1 được ước lượng, nếu đúng E2 được ước lượng và nó trở thành giá trị của biểu thức; nếu E1 sai, E2 được ước lượng và trở thành giá trị của biểu thức.

Ví dụ:

X = 10

Y = X > 9 ? 100 : 200

Thì Y được gán giá trị 100, nếu X nhỏ hơn 9 thì Y sẽ nhận giá trị là 200. Đoạn mã này tương đương cấu trúc if như sau:

X = 10

if (X < 9) Y = 100

else Y = 200

Toán tử con trỏ & và *

Một con trỏ là địa chỉ trong bộ nhớ của một biến. Một biến con trỏ là một biến được khai báo riêng để chứa một con trỏ đến một đối tượng của kiểu đã chỉ ra nó. Ta sẽ tìm hiểu kỹ hơn về con trỏ trong chương về

con trỏ. Ở đây, chúng ta sẽ đề cập ngắn gọn đến hai toán tử được sử dụng để thao tác với các con trỏ.

Toán tử thứ nhất là &, là một toán tử quy ước trả về địa chỉ bộ nhớ của hệ số của nó.

Ví dụ:m = &count

Đặt vào biến m địa chỉ bộ nhớ của biến count.

Chẳng hạn, biến count ở vị trí bộ nhớ 2000, giả sử count có giá trị là 100. Sau câu lệnh trên m sẽ nhận giá trị 2000.

Toán tử thứ hai là *, là một bổ sung cho &; đây là một toán tử quy ước trả về giá trị của biến được cấp phát tại địa chỉ theo sau đó.

Ví dụ:q = *m

Sẽ đặt giá trị của count vào q. Bây giờ q sẽ có giá trị là 100 vì 100 được lưu trữ tại địa chỉ 2000.

Toán tử dấu phẩy ,

Toán tử dấu , được sử dụng để kết hợp các biểu thức lại với nhau. Bên trái của toán tử dấu , luôn được xem là kiểu void. Điều đó có nghĩa là biểu thức bên phải trở thành giá trị của tổng các biểu thức được phân cách bởi dấu phẩy.

Ví dụ:x = (y=3,y+1);

Trước hết gán 3 cho y rồi gán 4 cho x. Cặp dấu ngoặc đơn là cần thiết vì toán tử dấu , có độ ưu tiên thấp hơn toán tử gán.

Xem các dấu ngoặc đơn và cặp dấu ngoặc vuông là toán tử

Trong C, cặp dấu ngoặc đơn là toán tử để tăng độ ưu tiên của các biểu thức bên trong nó.

Các cặp dấu ngoặc vuông thực hiện thao tác truy xuất phần tử trong mảng.

Tổng kết về độ ưu tiên

Cao nhất	() []
	! ~ ++ -- (Kiểu) * &
	* / %
	+ -
	<< >>
	< <= > >=
	&
	^
	|
	&&
	||
	?:

	= += -= *= /=
Thấp nhất	,

VI.2.9 Cách viết tắt trong C

Có nhiều phép gán khác nhau, đôi khi ta có thể sử dụng viết tắt trong C nữa. Chẳng hạn:

x = x + 10 được viết thànhx +=10

Toán tử += báo cho chương trình dịch biết để tăng giá trị của x lên 10.

Cách viết này làm việc trên tất cả các toán tử nhị phân (phép toán hai ngôi) của C. Tổng quát:

(Biến) = (Biến)(Toán tử)(Biểu thức)

có thể được viết:

(Biến)(Toán tử)=(Biểu thức)

CẤU TRÚC CỦA MỘT CHƯƠNG TRÌNH C

Tiền xử lý và biên dịch

Trong C, việc dịch (translation) một tập tin nguồn được tiến hành trên hai bước hoàn toàn độc lập với nhau:

- Tiền xử lý.

- Biên dịch.

Hai bước này trong phần lớn thời gian được nối tiếp với nhau một cách tự động theo cách thức mà ta có ấn tượng rằng nó đã được thực hiện như là một xử lý duy nhất. Nói chung, ta thường nói đến việc tồn tại của một

bộ tiền xử lý (preprocessor?) nhằm chỉ rõ chương trình thực hiện việc xử lý trước. Ngược lại, các thuật ngữ trình biên dịch hay sự biên dịch vẫn còn nhập nhằng bởi vì nó chỉ ra khi thì toàn bộ hai giai đoạn, khi thì lại là giai đoạn thứ hai.

Bước tiền xử lý tương ứng với việc cập nhật trong văn bản của chương trình nguồn, chủ yếu dựa trên việc diễn giải các mã lệnh rất đặc biệt gọi là các chỉ thị dẫn hướng của bộ tiền xử lý (destination directive of preprocessor); các chỉ thị này được nhận biết bởi chúng bắt đầu bằng ký hiệu (symbol) #.

Hai chỉ thị quan trọng nhất là:

- Chỉ thị sự gộp vào của các tập tin nguồn khác: #include

- Chỉ thị việc định nghĩa các macros hoặc ký hiệu: #define

Chỉ thị đầu tiên được sử dụng trước hết là nhằm gộp vào nội dung của các tập tin cần có (header file), không thể thiếu trong việc sử dụng một cách tốt nhất các hàm của thư viện chuẩn, phổ biến nhất là:

#include <stdio.h>

Chỉ thị thứ hai rất hay được sử dụng trong các tập tin thư viện (header file) đã được định nghĩa trước đó và thường được khai thác bởi các lập trình viên trong việc định nghĩa các ký hiệu như là:

#define NB_COUPS_MAX 100

#define SIZE 25

Cấu trúc một chương trình C

Một chương trình C bao gồm các phần như: Các chỉ thị tiền xử lý, khai báo biến ngoài, các hàm tự tạo, chương trình chính (hàm main).

Cấu trúc có thể như sau:

Các chỉ thị tiền xử lý (Preprocessor directives)#include <Tên tập tin thư viện>#define ….

Định nghĩa kiểu dữ liệu (phần này không bắt buộc): dùng để đặt tên lại cho một kiểu dữ liệu nào đó để gợi nhớ hay đặt 1 kiểu dữ liệu cho riêng mình dựa trên các kiểu dữ liệu đã có.Cú pháp: typedef <Tên kiểu cũ> <Tên kiểu mới>Ví dụ: typedef int SoNguyen; // Kiểu SoNguyen là kiểu int

Khai báo các prototype (tên hàm, các tham số, kiểu kết quả trả về,… của các hàm sẽ cài đặt trong phần sau, phần này không bắt buộc): phần này chỉ là các khai báo đầu hàm, không phải là phần định nghĩa hàm.

Khai báo các biến ngoài (các biến toàn cục) phần này không bắt buộc: phần này khai báo các biến toàn cục được sử dụng trong cả chương trình.

Chương trình chính phần này bắt buộc phải có<Kiểu dữ liệu trả về> main(){Các khai báo cục bộ trong hàm main: Các khai báo này chỉ tồn tại trong hàm mà thôi, có thể là khai báo biến hay khai báo kiểu.Các câu lệnh dùng để định nghĩa hàm mainreturn <kết quả trả về>; // Hàm phải trả về kết quả}

Cài đặt các hàm<Kiểu dữ liệu trả về> function1(các tham số){Các khai báo cục bộ trong hàm.Các câu lệnh dùng để định nghĩa hàmreturn <kết quả trả về>;}…

Lưu ý: Một số tập tin header thường dùng:

Một chương trình C bắt đầu thực thi từ hàm main (thông thường là từ câu lệnh đầu tiên đến câu lệnh cuối cùng).

Các tập tin thư viện thông dụng

Đây là các tập tin chứa các hàm thông dụng khi lập trinh C, muốn sử dụng các hàm trong các tập tin header này thì phải khai báo #include <Tên tập tin> ở phần đầu của chương trình

1) stdio.h: Tập tin định nghĩa các hàm vào/ra chuẩn (standard input/output). Gồm các hàm in dữ liệu (printf()), nhập giá trị cho biến (scanf()), nhận ký tự từ bàn phím (getc()), in ký tự ra màn hình (putc()), nhận một dãy ký tự từ bàm phím (gets()), in chuỗi ký tự ra màn hình (puts()), xóa vùng đệm bàn phím (fflush()), fopen(), fclose(), fread(), fwrite(), getchar(), putchar(), getw(), putw()…

2) conio.h : Tập tin định nghĩa các hàm vào ra trong chế độ DOS (DOS console). Gồm các hàm clrscr(), getch(), getche(), getpass(), cgets(), cputs(), putch(), clreol(),…

3) math.h: Tập tin định nghĩa các hàm tính toán gồm các hàm abs(), sqrt(), log(). log10(), sin(), cos(), tan(), acos(), asin(), atan(), pow(), exp(),…

4) alloc.h: Tập tin định nghĩa các hàm liên quan đến việc quản lý bộ nhớ. Gồm các hàm calloc(), realloc(), malloc(), free(), farmalloc(), farcalloc(), farfree(), …

5) io.h: Tập tin định nghĩa các hàm vào ra cấp thấp. Gồm các hàm open(), _open(), read(), _read(), close(), _close(), creat(), _creat(), creatnew(), eof(), filelength(), lock(),…

6) graphics.h: Tập tin định nghĩacác hàm liên quan đến đồ họa. Gồm initgraph(), line(), circle(), putpixel(), getpixel(), setcolor(), …

Còn nhiều tập tin khác nữa.

Cú pháp khai báo các phần bên trong môt chương trình C

Chỉ thị #include để sử dụng tập tin thư viện

Cú pháp:

#include <Tên tập tin> // Tên tập tin được đạt trong dấu <>

hay #include “Tên đường dẫn”

Menu Option của Turbo C có mục INCLUDE DIRECTORIES, mục này dùng để chỉ định các tập tin thư viện được lưu trữ trong thư mục nào.

Nếu ta dùng #include<Tên tập tin> thì Turbo C sẽ tìm tập tin thư viện trong thư mục đã được xác định trong INCLUDE DIRECTORIES.

Ví dụ: include <stdio.h>

Nếu ta dùng #include”Tên đường dẫn” thì ta phải chỉ rõ tên ở đâu, tên thư mục và tập tin thư viện.

Ví dụ:#include”C:\TC\math.h”

Trong trường hợp tập tin thư viện nằm trong thư mục hiện hành thì ta chỉ cần đưa tên tập tin thư viện. Ví dụ: #include”math.h”.

Ví dụ:

#include <stdio.h>

#include <conio.h>

#include “math.h”

Chỉ thị #define để định nghĩa hằng số

Cú pháp:

#define <Tên hằng> <Giá trị>

Ví dụ:

#define MAXINT 32767

Khai báo các prototype của hàm

Cú pháp:

<Kiểu kết quả trả về> Tên hàm (danh sách đối số)

Ví dụ:

long giaithua(int n); //Hàm tính giai thừa của số nguyên n

double x_mu_y(float x, float y);/*Hàm tính x mũ y*/

Cấu trúc của hàm "bình thường"

Cú pháp:

<Kiểu kết quả trả về> Tên hàm (các đối số)

{

Các khai báo và các câu lệnh định nghĩa hàm

return kết quả;

}

Ví dụ:

int tong(int x, int y) /*Hàm tính tổng 2 số nguyên*/

{

return (x+y);

```
}

float tong(float x, float y) /*Hàm tính tổng 2 số thực*/

{

return (x+y);

}
```

Cấu trúc của hàm main

Hàm main chính là chương trình chính, gồm các lệnh xử lý, các lời gọi các hàm khác.

Cú pháp:

```
<Kết quả trả về> main( đối số)

{

Các khai báo và các câu lệnh định nghĩa hàm

return <kết quả>;

}
```

Ví dụ 1:

```
int main()

{

printf("Day la chuong trinh chinh");

getch();

return 0;
```

```
}
```

Ví dụ 2:

```
int main()
{
int a=5, b=6,c;
float x=3.5, y=4.5,z;
printf("Day la chuong trinh chinh");
c=tong(a,b);
printf("\n Tong cua %d va %d la %d",a,b,c);
z=tong(x,y);
printf("\n Tong cua %f và %f là %f", x,y,z);
getch();
return 0;
}
```

BÀI TẬP

Bài 1: Biểu diễn các hằng số nguyên 2 byte sau đây dưới dạng số nhị phân, bát phân, thập lục phân

a)12b) 255c) 31000d) 32767e) -32768

Bài 2: Biểu diễn các hằng ký tự sau đây dưới dạng số nhị phân, bát phân.

a) ‘A’b) ’a’c) ‘Z’d) ’z’

Giới thiệu về ngôn ngữ C và môi trường turbo C 3.0
Học xong chương này, sinh viên sẽ nắm được các vấn đề sau: - Tổng quan về ngôn ngữ lập trình C. - Môi trường làm việc và cách sử dụng Turbo C 3.0.

TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH C

C là ngôn ngữ lập trình cấp cao, được sử dụng rất phổ biến để lập trình hệ thống cùng với Assembler và phát triển các ứng dụng.

Vào những năm cuối thập kỷ 60 đầu thập kỷ 70 của thế kỷ XX, Dennish Ritchie (làm việc tại phòng thí nghiệm Bell) đã phát triển ngôn ngữ lập trình C dựa trên ngôn ngữ BCPL (do Martin Richards đưa ra vào năm 1967) và ngôn ngữ B (do Ken Thompson phát triển từ ngôn ngữ BCPL vào năm 1970 khi viết hệ điều hành UNIX đầu tiên trên máy PDP-7) và được cài đặt lần đầu tiên trên hệ điều hành UNIX của máy DEC PDP-11.

Năm 1978, Dennish Ritchie và B.W Kernighan đã cho xuất bản quyển "Ngôn ngữ lập trình C" và được phổ biến rộng rãi đến nay.

Lúc ban đầu, C được thiết kế nhằm lập trình trong môi trường của hệ điều hành Unix nhằm mục đích hỗ trợ cho các công việc lập trình phức tạp. Nhưng về sau, với những nhu cầu phát triển ngày một tăng của công việc lập trình, C đã vượt qua khuôn khổ của phòng thí nghiệm Bell và nhanh chóng hội nhập vào thế giới lập trình để rồi các công ty lập trình sử dụng một cách rộng rãi. Sau đó, các công ty sản xuất phần mềm lần lượt đưa ra các phiên bản hỗ trợ cho việc lập trình bằng ngôn ngữ C và chuẩn ANSI C cũng được khai sinh từ đó.

Ngôn ngữ lập trình C là một ngôn ngữ lập trình hệ thống rất mạnh và rất "mềm dẻo", có một thư viện gồm rất nhiều các hàm (function) đã được tạo sẵn. Người lập trình có thể tận dụng các hàm này để giải quyết các bài toán mà không cần phải tạo mới. Hơn thế nữa, ngôn ngữ C hỗ trợ rất nhiều phép toán nên phù hợp cho việc giải quyết các bài toán kỹ thuật có nhiều công thức phức tạp. Ngoài ra, C cũng cho phép người lập trình tự định nghĩa thêm các kiểu dữ liệu trừu tượng khác. Tuy nhiên, điều mà

người mới vừa học lập trình C thường gặp “rắc rối” là “hơi khó hiểu” do sự “mềm dẻo” của C. Dù vậy, C được phổ biến khá rộng rãi và đã trở thành một công cụ lập trình khá mạnh, được sử dụng như là một ngôn ngữ lập trình chủ yếu trong việc xây dựng những phần mềm hiện nay.

Ngôn ngữ C có những đặc điểm cơ bản sau:

- Tính cô đọng (compact): C chỉ có 32 từ khóa chuẩn và 40 toán tử chuẩn, nhưng hầu hết đều được biểu diễn bằng những chuỗi ký tự ngắn gọn.
- Tính cấu trúc (structured): C có một tập hợp những chỉ thị của lập trình như cấu trúc lựa chọn, lặp… Từ đó các chương trình viết bằng C được tổ chức rõ ràng, dễ hiểu.
- Tính tương thích (compatible): C có bộ tiền xử lý và một thư viện chuẩn vô cùng phong phú nên khi chuyển từ máy tính này sang máy tính khác các chương trình viết bằng C vẫn hoàn toàn tương thích.
- Tính linh động (flexible): C là một ngôn ngữ rất uyển chuyển và cú pháp, chấp nhận nhiều cách thể hiện, có thể thu gọn kích thước của các mã lệnh làm chương trình chạy nhanh hơn.
- Biên dịch (compile): C cho phép biên dịch nhiều tập tin chương trình riêng rẽ thành các tập tin đối tượng (object) và liên kết (link) các đối tượng đó lại với nhau thành một chương trình có thể thực thi được (executable) thống nhất.

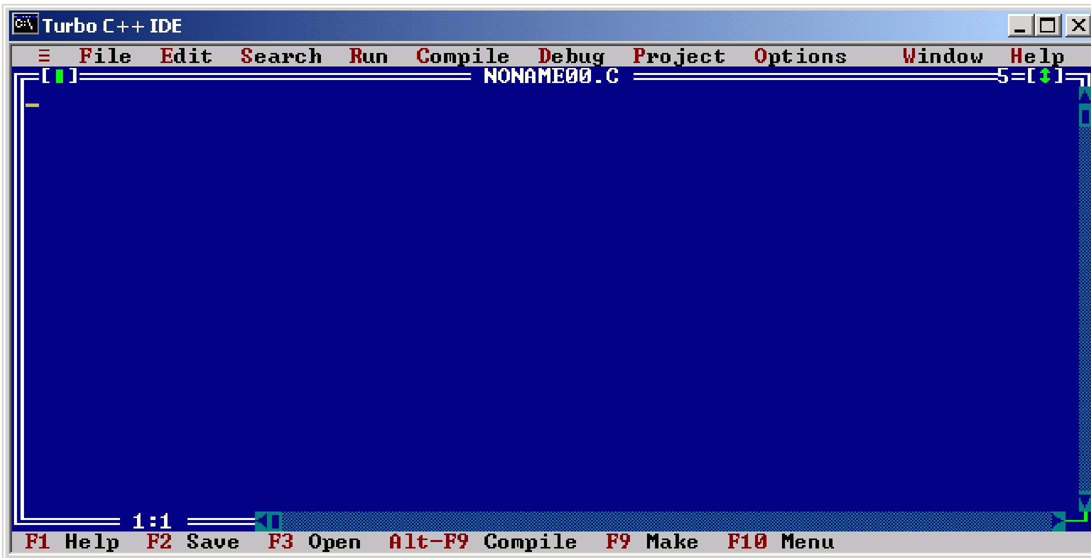
MÔI TRƯỜNG LẬP TRÌNH TURBO C

Turbo C là môi trường hỗ trợ lập trình C do hãng Borland cung cấp. Môi trường này cung cấp các chức năng như: soạn thảo chương trình, dịch, thực thi chương trình… Phiên bản được sử dụng ở đây là Turbo C 3.0.

Gọi Turbo C

Chạy Turbo C cũng giống như chạy các chương trình khác trong môi trường DOS hay Windows, màn hình sẽ xuất hiện menu của Turbo C có

dạng như sau:



Dòng trên cùng gọi là thanh menu (menu bar). Mỗi mục trên thanh menu lại có thể có nhiều mục con nằm trong một menu kéo xuống.

Dòng dưới cùng ghi chức năng của một số phím đặc biệt. Chẳng hạn khi gõ phím F1 thì ta có được một hệ thống trợ giúp mà ta có thể tham khảo nhiều thông tin bổ ích.

Muốn vào thanh menu ngang ta gõ phím F10. Sau đó dùng các phím mũi tên qua trái hoặc phải để di chuyển vùng sáng tới mục cần chọn rồi gõ phím Enter. Trong menu kéo xuống ta lại dùng các phím mũi tên lên xuống để di chuyển vùng sáng tới mục cần chọn rồi gõ Enter.

Ta cũng có thể chọn một mục trên thanh menu bằng cách giữ phím Alt và gõ vào một ký tự đại diện của mục đó (ký tự có màu sắc khác với các ký tự khác). Chẳng hạn để chọn mục File ta gõ Alt-F (F là ký tự đại diện của File)

Soạn thảo chương trình mới

Muốn soạn thảo một chương trình mới ta chọn mục New trong menu File (File ->New)

Trên màn hình sẽ xuất hiện một vùng trống để cho ta soạn thảo nội dung của chương trình. Trong quá trình soạn thảo chương trình ta có thể sử dụng các phím sau:

Các phím xem thông tin trợ giúp:

- F1: Xem toàn bộ thông tin trong phần trợ giúp.

- Ctrl-F1: Trợ giúp theo ngữ cảnh (tức là khi con trỏ đang ở trong một từ nào đo, chẳng hạn int mà bạn gõ phím Ctrl-F1 thì bạn sẽ có được các thông tin về kiểu dữ liệu int)

Các phím di chuyển con trỏ trong vùng soạn thảo chương trình:

Phím	Ý nghĩa	Phím tắt (tổ hợp phím)
Enter	Đưa con trỏ xuống dòng	
Mũi tên đi lên	Đưa con trỏ lên hàng trước	Ctrl-E
Mũi tên đi xuống	Đưa con trỏ xuống hàng sau	Ctrl-X
Mũi tên sang trái	Đưa con trỏ sang trái một ký tự	Ctrl-S
Mũi tên sang	Đưa con trỏ sang phải	Ctrl-D

phải	một ký tự	
End	Đưa con trỏ đến cuối dòng	
Home	Đưa con trỏ đến đầu dòng	
PgUp	Đưa con trỏ lên trang trước	Ctrl-R
PgDn	Đưa con trỏ xuống trang sau	Ctrl-C
	Đưa con trỏ sang từ bên trái	Ctrl-A
	Đưa con trỏ sang từ bên phải	Ctrl-F

Các phím xoá ký tự/ dòng:

Phím	Ý nghĩa	Phím tắt
Delete	Xoá ký tự tại ví trí con trỏ	Ctrl-G
BackSpace	Di chuyển sang trái đồng thời xoá ký tự đứng trước con trỏ	Ctrl-H

	Xoá một dòng chứa con trỏ	Ctrl-Y
	Xóa từ vị trí con trỏ đến cuối dòng	Ctrl-Q-Y
	Xóa ký tự bên phải con trỏ	Ctrl-T

Các phím chèn ký tự/ dòng:

Insert	Thay đổi viết xen hay viết chồng
Ctrl-N	Xen một dòng trống vào trước vị trí con trỏ

Sử dụng khối :

Khối là một đoạn văn bản chương trình hình chữ nhật được xác định bởi đầu khối là góc trên bên trái và cuối khối là góc dưới bên phải của hình chữ nhật. Khi một khối đã được xác định (trên màn hình khối có màu sắc khác chỗ bình thường) thì ta có thể chép khối, di chuyển khối, xoá khối... Sử dụng khối cho phép chúng ta soạn thảo chương trình một cách nhanh chóng. sau đây là các thao tác trên khối:

Phím tắt	Ý nghĩa

Ctrl-K-B	Đánh dấu đầu khối
Ctrl-K-K	Đánh dấu cuối khối
Ctrl-K-C	Chép khối vào sau vị trí con trỏ
Ctrl-K-V	Chuyển khối tới sau vị trí con trỏ
Ctrl-K-Y	Xoá khối
Ctrl-K-W	Ghi khối vào đĩa như một tập tin
Ctrl-K-R	Đọc khối (tập tin) từ đĩa vào sau vị trí con trỏ
Ctrl-K-H	Tắt/mở khối
Ctrl-K-T	Đánh dấu từ chứa chon trỏ
Ctrl-K-P	In một khối

Các phím, phím tắt thực hiện các thao tác khác:

Phím	Ý nghĩa	Phím tắt
F10	Kích hoạt menu chính	Ctrl-K-D, Ctrl-K-Q
F2	Lưu chương trình đang soạn vào đĩa	Ctrl-K-S
F3	Tạo tập tin mới	

Tab	Di chuyển con trỏ một khoảng đồng thời đẩy dòng văn bản	Ctrl-I
ESC	Hủy bỏ thao tác lệnh	Ctrl-U
	Đóng tập tin hiện tại	Alt-F3
	Hiện hộp thoại tìm kiếm	Ctrl-Q-F
	Hiện hộp thoại tìm kiếm và thay thế	Ctrl-Q-A
	Tìm kiếm tiếp tục	Ctrl-L

Ví dụ: Bạn hãy gõ đoạn chương trình sau:

```
#include <stdio.h>

#include<conio.h>

int main ()

{

char ten[50];

printf(“Xin cho biet ten cua ban !”);

scanf(“%s”,ten);

printf(“Xin chao ban %s”,ten);

getch();

return 0;

}
```

Ghi chương trình đang soạn thảo vào đĩa

Sử dụng File/Save hoặc gõ phím F2. Có hai trường hợp xảy ra:

- Nếu chương trình chưa được ghi lần nào thì một hội thoại sẽ xuất hiện cho phép bạn xác định tên tập tin (FileName). Tên tập tin phải tuân thủ quy cách đặt tên của DOS và không cần có phần mở rộng (sẽ tự động có phần mở rộng là .C hoặc .CPP sẽ nói thêm trong phần Option). Sau đó gõ phím Enter.

- Nếu chương trình đã được ghi một lần rồi thì nó sẽ ghi những thay đổi bổ sung lên tập tin chương trình cũ.

Chú ý: Để đề phòng mất điện trong khi soạn thảo chương trinh thỉnh thoảng bạn nên gõ phím F2.

Quy tắc đặt tên tập tin của DOS: Tên của tập tin gồm 2 phần: Phần tên và phần mở rộng.

- Phần tên của tập tin phải bắt đầu là 1 ký tự từ a..z (không phân biệt hoa thường), theo sau có thể là các ký tự từ a..z, các ký số từ 0..9 hay dấu gạch dưới (_), phần này dài tối đa là 8 ký tự.
- Phần mở rộng: phần này dài tối đa 3 ký tự.

Ví dụ: Ghi chương trình vừa soạn thảo trên lên đĩa với tên là CHAO.C

Thực hiện chương trình

Để thực hiện chương trình hãy dùng Ctrl-F9 (giữ phím Ctrl và gõ phím F9).

Ví dụ: Thực hiện chương trình vừa soạn thảo xong và quan sát trên màn hình để thấy kết quả của việc thực thi chương trình sau đó gõ phím bất kỳ để trở lại với Turbo.

Mở một chương trình đã có trên đĩa

Với một chương trình đã có trên đĩa, ta có thể mở nó ra để thực hiện hoặc sửa chữa bổ sung. Để mở một chương trình ta dùng File/Open hoặc gõ phím F3. Sau đó gõ tên tập tin vào hộp File Name hoặc lựa chọn tập tin trong danh sách các tập tin rồi gõ Enter.

Ví dụ: Mở tập tin CHAO.C sau đó bổ sung để có chương trình mới như sau:

```
#include <stdio.h>

#include<conio.h>

int main ()

{

char ten[50];

printf("Xin cho biet ten cua ban !");

scanf("%s",ten);

printf("Xin chao ban %s\n ",ten);

printf("Chao mung ban den voi Ngon ngu lap trinh C");

getch();

return 0;

}
```

Ghi lại chương trình này (F2) và cho thực hiện (Ctrl-F9). Hãy so sánh xem có gì khác trước?

Thoát khỏi Turbo C và trở về DOS (hay Windows)

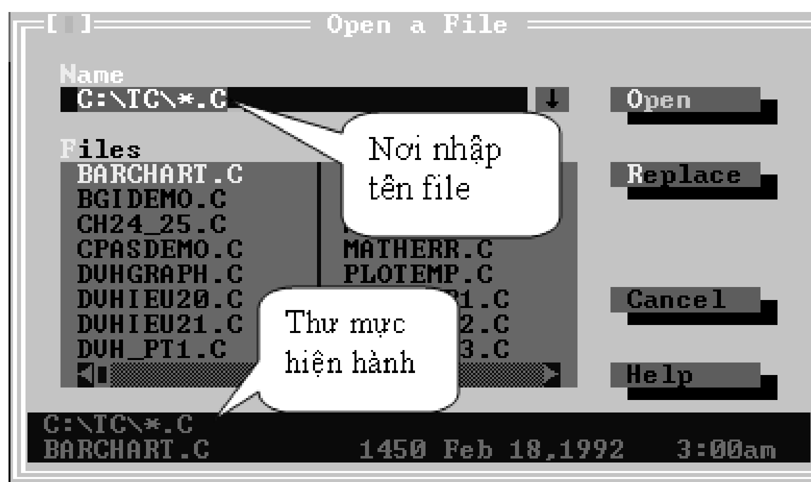
Dùng File/Exit hoặc Alt-X.

Sử dụng một số lệnh trên thanh menu

Các lệnh trên menu File (Alt -F)

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED. - Lệnh New : Dùng để tạo mới một chương trình. Tên ngầm định của chương trình là NONAMEXX.C (XX là 2 số từ 00 đến 99).

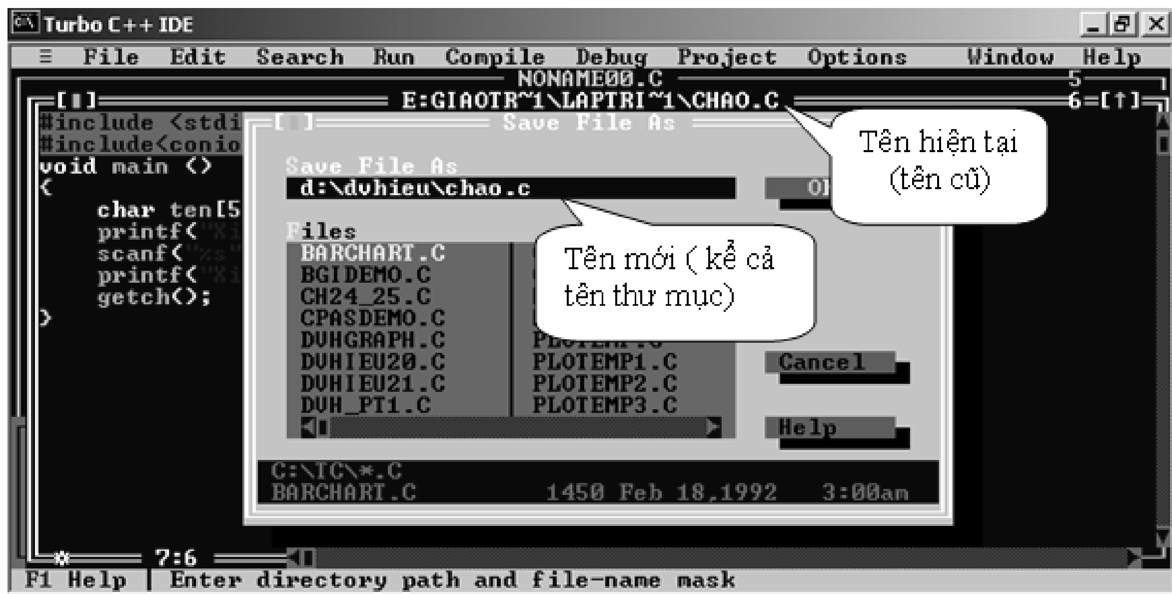
- Lệnh Open : Dùng để mở một chương trình đã có sẵn trên đĩa để sửa chữa, bổ sung hoặc để thực hiện chương trình đó. Khi tập tin được mở thì văn bản chương trình được trình bày trong vùng soạn thảo; hộp thoại Open như sau:



Trong trường hợp ta nhập vào tên tập tin chưa tồn tại thì chương trình được tạo mới và sau này khi ta lưu trữ, chương trình được lưu với tên đó.

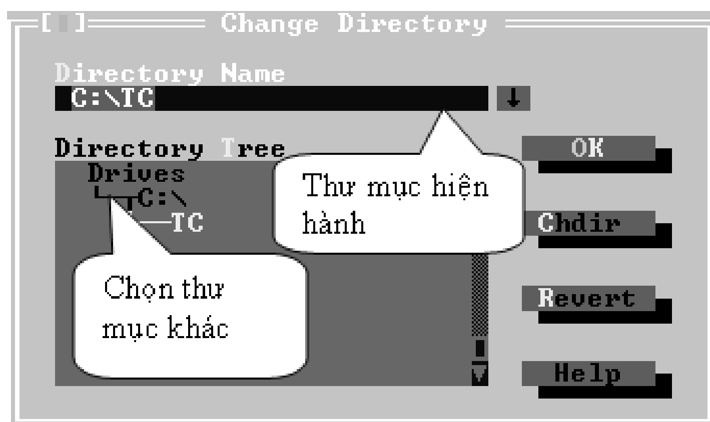
- Lệnh Save : Dùng để lưu chương trình đang soạn thảo vào đĩa.

- Lệnh Save as... : Dùng để lưu chương trình đang soạn thảo với tên khác, hộp thoại lưu tập tin đang soạn với tên khác như sau:



- Lệnh : Save All: Trong lúc làm việc với Turbo C, ta có thể mở một lúc nhiều chương trình để sửa chữa, bổ sung. Lệnh Save All dùng để lưu lại mọi thay đổi trên tất cả các chương trình đang mở ấy..

- Lệnh Change Dir ... : Dùng để đổi thư mục hiện hành



- Lệnh Print : Dùng để in chương trình đang soạn thảo ra máy in.

- Lệnh Printer Setup ...: Dùng để thiết đặt một số thông số cho máy in.

- Lệnh Dos Shell : Dùng để thoát tạm thời về Dos, để trở lại Turbo C ta đánh EXIT.

- Lệnh Exit : Dùng để thoát khỏi C.

Các lệnh trên menu Edit (Alt -E)

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED. - Lệnh Undo : Dùng để hủy bỏ thao tác soạn thảo cuối cùng trên cửa số soạn thảo.

- Lệnh Redo : Dùng để phục hồi lại thao tác đã bị Undo cuối cùng.

- Lệnh Cut : Dùng để xóa một phần văn bản đã được đánh dấu khối, phần dữ liệu bị xóa sẽ được lưu vào một vùng nhớ đặc biệt gọi là Clipboard.

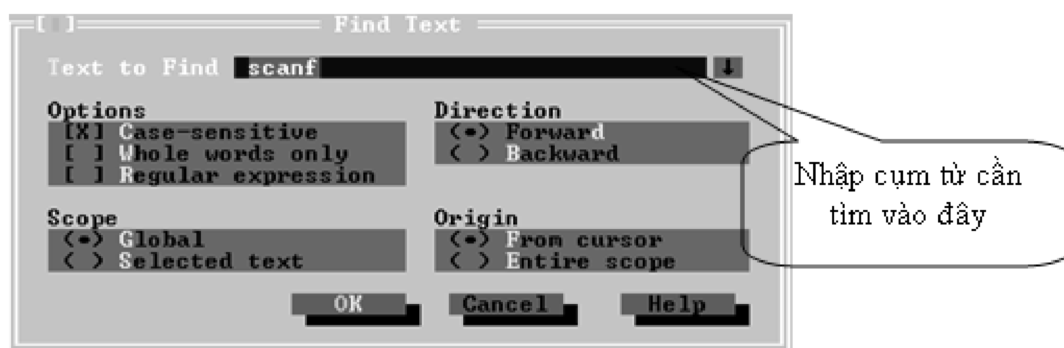
- Lệnh Copy : Dùng để chép phần chương trình đã được đánh dấu khối vào Clipboard.

- Lệnh Paste : Dùng để dán phần chương trình đang được lưu trong Clipboard vào của số đang soạn thảo, bắt đầu tại vị trí của con trỏ.

- Lệnh Clear : Dùng để xóa phần dữ liệu đã được đánh dấu khối, dữ liệu bị xóa không được lưu vào Clipboard.

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED. - Lệnh Show clipboard : Dùng để hiển thị phần chương trình đang được lưu trong Clipboard trong một cửa sổ mới.

Các lệnh trên menu Search (Alt -S)

- Lệnh Find ...: Dùng để tìm kiếm một cụm từ trong văn bản chương trình. Nếu tìm thấy thì con trỏ sẽ di chuyển đến đoạn văn bản trùng với cụm từ cần tìm; hộp thoại Find như sau:
- 

Ý nghĩa các lựa chọn trong hộp thoại trên như sau:

- Case sentitive : Phân biệt chữ IN HOA với chữ in thường trong khi so sánh cụm từ cần tìm với văn bản chương trình.
- Whole word only: Một đoạn văn bản chương trình trùng với toàn bộ cụm từ cần tìm thì mới được xem là tìm thấy.
- Regular expression: Tìm theo biểu thức
- Global:Tìm trên tất cả tập tin.
- Forward : Tìm đến cuối tập tin.
- Selected text: Chỉ tìm trong khối văn bản đã được đánh dấu.
- Backward: Tìm đến đầu tập tin.
- From cursor : Bắt đầu từ vị trí con nháy.
- Entire scope: Bắt đầu tại vị trí đầu tiên của khối hoặc tập tin.

- Lệnh Replace...: Dùng để tìm kiếm một đoạn văn bản nào đó, và tự động thay bằng một đoạn văn bản khác, hộp thoại replace như sau:

Tìm các cụm từ Scanf và thay thế bằng scanf ***SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.***

- Lệnh Search again : Dùng để thực hiện lại việc tìm kiếm.

- Các lệnh còn lại trên menu Search, các bạn sẽ tìm hiểu thêm khi thực hành trực tiếp trên máy tính.

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED. Các lệnh trên menu Run (Alt -R)

- Lệnh Run : Dùng để thực thi hay "chạy" một chương trình.

- Lệnh Step over : Dùng để "chạy" chương trình từng bước.

- Lệnh Trace into : Dùng để chạy chương trình từng bước. Khác với lệnh Step over ở chỗ: Lệnh Step over không cho chúng ta xem từng bước "chạy" trong chương trình con, còn lệnh Trace into cho chúng ta xem từng bước trong chương trình con.

- Các lệnh còn lại, các bạn sẽ tìm hiểu thêm khi thực hành trên máy.

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED. Các lệnh trên menu Compile (Alt C)

- Lệnh Complie: Biên dịch một chương trình.

- Lệnh Make , Build, … : Các lệnh này bạn sẽ tìm hiểu thêm khi thực hành trực tiếp trên máy tính.

- Lệnh Information : Dùng để hiện thị các thông tin về chương trình, Mode, môi trường .

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED. Các lệnh trên menu Debug (Alt-D)

Trên menu Debug bao gồm một số lệnh giúp người lập trình "gỡ rối" chương trình . Người lập trình sử dụng chức năng "gỡ rối" khi gặp một số "lỗi" về thuật toán, sử dụng biến nhớ…

- Lệnh Breakpoints: Dùng để đặt "điểm dừng" trong chương trình. Khi chương trình thực thi đến "điểm dừng thì nó sẽ dừng lại" .

- Lệnh Watch : Dùng để mở một cửa sổ hiển thị kết quả trung gian của một biến nhớ nào đó khi chạy chương trình từng bước.

- Lệnh Evaluate/Modify: Bạn sẽ tìm hiểu khi thực hành trực tiếp trên máy.

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED. Các lệnh trên menu Project (Alt- P)

Trên menu Project bao gồm các lệnh liên quan đến dự án như : đóng, mở, thêm , xóa các mục,…

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED. Các lệnh trên menu Option (Alt -O)

Trên menu Option bao gồm các lệnh giúp người lập trình thiết đặt một số tự chọn khi chạy chương trình. Thông thường, người lập trình không cần phải thiết đặt lại các tự chọn.

- Lệnh Compiler ...: Dùng để thiết đặt lại một số thông số khi biên dịch chương trình như hình sau

Phần trình bày dưới đây thuộc về 3 mục: Directories, Enviroment và Save; các phần khác sinh viên tự tìm hiểu.

- Lệnh Directories...: Dùng để đặt lại đường dẫn tìm đến các tập tin cần thiết khi biên dịch chương trình như hình sau:

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

- Include directory: Thư mục chứa các tập tin mà chúng ta muốn đưa vào chương trình (các tập tin .h trong dòng #include).
- Library directory : Thư mục chứa các tập tin thư viện (các tập tin .Lib)
- Output directory: Thư mục chứa các tập tin "đối tượng " (có phần mở rộng là .OBJ), tập tin thực thi (.exe) khi biên dịch chương trình.
- Source directory: Thư mục chứa các tập tin "nguồn" (có phần mở rộng là .obj, .lib).

- Lệnh Environment: dùng để thiết lập môi trường làm việc như:

- Reference…: Các tham chiếu.
- Editor: Môi trường soạn thảo gồm: tạo tập tin dự phòng khi có sự chỉnh sửa (create backup file), chế độ viết đè (insert mode), tự động thụt đầu dòng (indent), đổi màu từ khóa (Syntax highlighting)… Đặc biệt, trong phần này là thiết lập phần mở rộng mặc định (Default Extension) của tập tin chương trình là C hay CPP (C Plus Plus: C++).

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

- Mouse...: Đặt chuột.
- Colors…: Đặt màu.

Các lệnh trên menu Window (Alt- W)

Trên menu Window bao gồm các lệnh thao tác đến cửa sổ như:

- Lệnh Cascade : Dùng để sắp xếp các cửa sổ.

- Lệnh Close all : Dùng để đóng tất cả các cửa sổ.

- Lệnh Zoom: Dùng để phóng to/ thu nhỏ cửa sổ.

- Các lệnh Tile, Refresh display, Size/ Move, Next, Previous, Close, List...: Các bạn sẽ tìm hiểu thêm khi thực hành trực tiếp trên máy tính.

II.7.10. Các lệnh trên menu Help (Alt- H)

Trên menu Help bao gồm các lệnh gọi trợ giúp khi người lập trình cần giúp đỡ một số vấn đề nào đó như: Cú pháp câu lệnh, cách sử dụng các hàm có sẵn…

- Lệnh Contents: Hiện thị toàn bộ nội dung của phần help.

- Lệnh Index : Hiển thị bảng tìm kiếm theo chỉ mục.

- Các lệnh còn lại, bạn sẽ tìm hiểu khi thực hành trên máy.